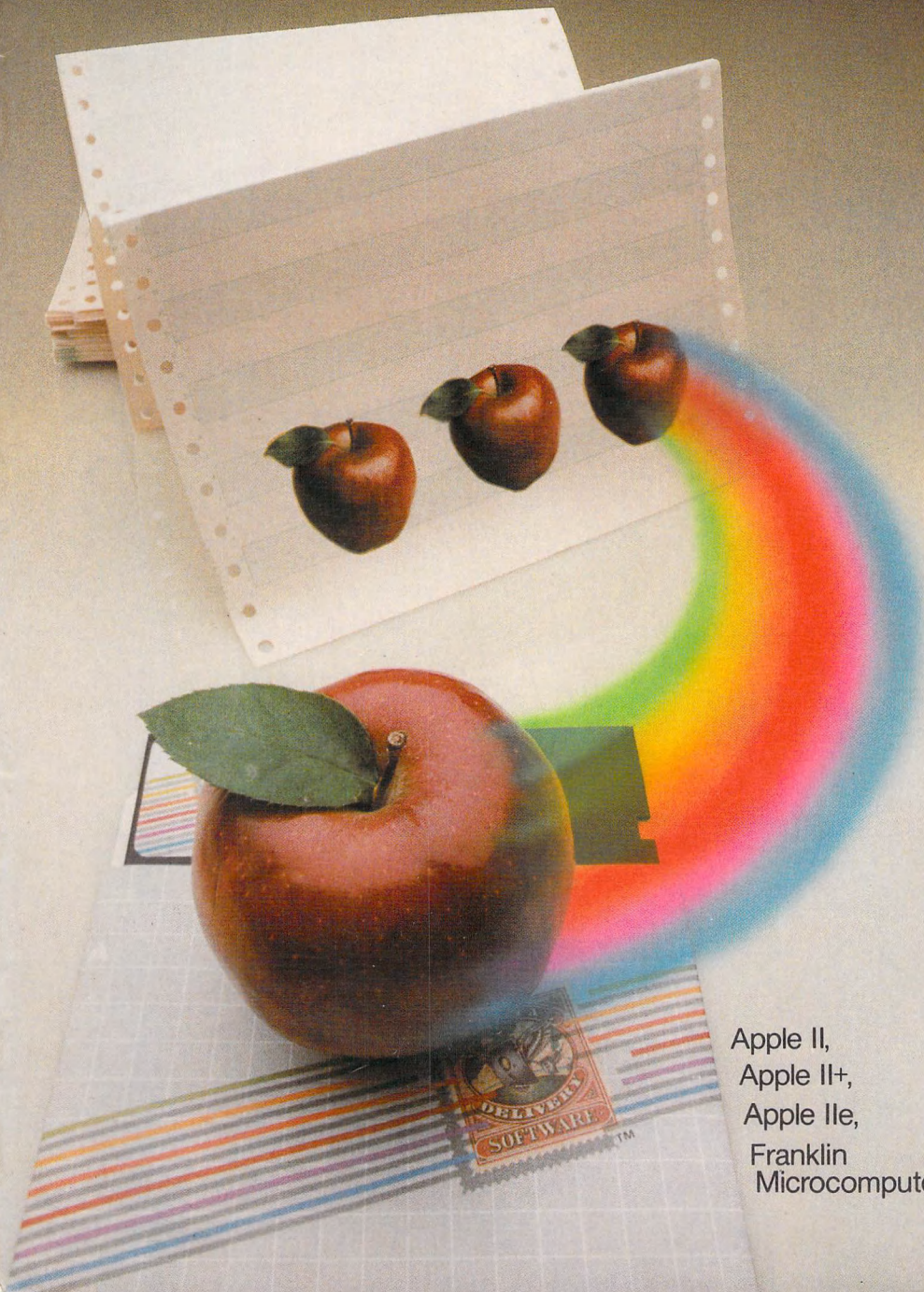
DISKS, FILES, AND PRINTERS FOR THE APPLE® II

BRIAN D. BLACKWOOD AND GEORGE H. BLACKWOOD



Apple II,
Apple II+,
Apple IIe,
Franklin
Microcomputers

Disks, Files, and Printers for the Apple® II

Dr. George H. Blackwood is a retired Navy pilot and a former college professor with bachelor's, master's, education specialist, and D.D.S. degrees. He now devotes full time to writing.

Brian D. Blackwood has studied computer science and engineering at Michigan State University, and has a B.S. degree in computer science from Lamar University. He is presently employed as a programmer at a large data processing center that services banks and financial institutions.

Disks, Files, and Printers for the Apple® II

By
Brian D. Blackwood
&
George H. Blackwood

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

FIRST EDITION
FIRST PRINTING—1983

International Standard Book Number: 0-672-22163-2
Library of Congress Catalog Card Number: 83-61068

Edited by: *C. Herbert Feltner*
Illustrated by: *Jill E. Martin*

Printed in the United States of America.

Preface

Disks, Files, and Printers for the Apple® II is written for students, of all ages, who are interested in studying the first level of list processing, or files. Files are composed of individual records, and the records are composed of individual fields.

Most files are stored on an external storage medium, outside of computer memory. The external storage memory consists primarily of magnetic tape, or disks.

Before 1950, the primary source of external storage was magnetic tape. Magnetic tape moves in a linear direction across a tape read/write head, and can move in only two directions, forward and backward. The linear, bidirectional, motion of tape required that records be placed one after the other, or sequentially. If the first record in the file was read, and the next record to be read was the last record in the file, it took a long time to get from the first record to the last record. To get from the first record to the last record, the tape had to run sequentially from beginning to end. This took a relatively long time. The time to get from one record to the next record is called the access time. The access time of a sequential access record file is slow.

After 1950, the circular drum and later the circular disk were introduced as a means of external memory storage. The disk rotated, and the read/write head moved from the center of the disk to the outside of the disk, and vice versa. The circular shape and the disk rotation allowed a second type of access to be used in file manipulation. The files placed on disk could be

accessed in a random fashion. On a circular file, the first record in the file might be next to the last record in the file. To go from the first record in the file, to the last record, in a random access file, took a relatively short time. The access time in a random access file is very fast.

Records in a sequential file can be any length, but records in a random file must be of fixed length. Fixed length records are one of the disadvantages of a random access file, because they waste storage space. The random access file trades wasted storage space for reduced access time.

The programming structure is entirely different for the two access methods. The sequential access method uses a beginning of record marker, and an end of record marker, to delineate one record from the next. The file access time is determined by how fast the tape can move across the read/write head.

In programming random access files, each record must be of fixed length. Each field must be in proper relationship with the other fields. A pointer, that points to the data in the record, can be used to link the records. The pointer and the data in the record must be less than or equal to the fixed length of the record. The pointer relates the preceding record, and the succeeding record. If a new record is placed in the file, it must be linked in the correct relationship between the preceding record and the succeeding record.

There are numerous techniques to handle the programming structure of random access files. In this book, only the linked list technique is presented.

In the linked list technique, a record contains two different structures, a pointer, and an item. The pointer is a variable containing a specific value. An item is the data contained in the fields of a record.

The linked list technique uses a pointer to point to the next record. The pointer contains a value that is linked to a specific record in the file. When a record is added, or deleted, from the file, the linked pointer value is changed.

A record stored on disk must be printed in hardcopy (on paper) form to be of general use. Thus, the printer is the final destination of the file.

The Apple II computer, 40 column screen, has two functions, TAB and SPC (space), to control column output to the screen. Most printers have at least an 80 column output capability.

The SPC function can be used to the 80 column capability of the printer. However, it is time consuming to program every case to control the output. The TAB function limits the printer to 40 columns.

Chapter 13 offers two programs to overcome the deficiencies of the TAB and SPC functions on the 40 column Apple II computer.

Disks, Files, and Printers for the Apple II is the fourth in a series of books specifically written for the beginning programmer using the Apple II computer. It is offered to augment the information presented in *Intimate Instructions in Integer Basic, Applesoft Language,* and *Apple Fortran.* The four books are intended to form a library for the beginning programmer. The missing link in the series is Pascal language, and there are several good books that cover Pascal.

BRIAN D. BLACKWOOD
GEORGE H. BLACKWOOD

Contents

CHAPTER 11

CHAPTER 12

CHAPTER 13

APPENDIX

chapter 1

Introduction to Disks

The simplest computer system consists of a computer and a video display module, often referred to as a VDM or a CRT. With this system, the input comes from the keyboard, and the output goes to the CRT. When a cassette tape recorder is added to the system, an additional source of input and output is added. Input can come from the keyboard or tape, and output can go to the CRT or tape.

When the user becomes more sophisticated in the use of the computer system, disks and a printer are added. The input can then come from keyboard, tape, or disk, and the output can go to CRT, tape, disk, or printer. With flexibility comes complexity.

Input must be specified as coming from a source, such as the keyboard, tape, or disk. Output must be specified as to a destination, such as the CRT, tape, disk, or printer.

As the computer system develops from a simple system to the complex system, the commands to control the system increase in number and complexity.

CHAPTER 1 HOLDS A LIBRARY OF FUNCTIONS

Most of the statements and commands used with the complex system are given in Chapter 1, Tables 1-1, 1-2, and 1-3, for three reasons.

Table 1-1. System Commands for the Computer, CRT, and Cassette Tape Recorder

Unit	Type	Command	Referenced By:
Keyboard	Input	Input	Set by the monitor
CRT	Output	Print	Set by the monitor
Cassette	I/O	None	Special software
Cassette	Input	Load	Special software
Cassette	Output	Save	Special software

Table 1-2. System Commands for the Computer, CRT, Cassette, Disks, and Printer

Unit	Type	Command	Referenced By:
Keyboard	Input	IN#0	Set by DOS control
CRT	Output	PR#0 PRINT	Set by DOS control
Cassette	I/O	None	Special software
Disk	Input	Read Input	Set by DOS control
Disk	Output	Write Print	Set by DOS control
Printer	Output	PR#1 I$, Q$	Set by DOS control

1. To alert the reader to the number and complexity of the statements and commands.
2. To give the reader an overview of the relationship of commands that are used with disks, files, and printers.
3. To give the reader a quick reference library of commands to permit maximum use of the system as soon as possible. The commands are in one location so the reader does not have to search through several pages of text to find out how to use the complex system.

The primary thrust of this book is to teach the user how to create and use text files while running a computer program, and to make hardcopies of the results on the printer. To accomplish this goal, a logical sequence of events should occur.

1. The disk operating system must be activated, and disks initialized, so they can be used with the disk operating system—Chapter 3.
2. Programs will be saved to disk. The disk is cataloged to list the file names contained on the disk to the CRT, and the programs will be loaded from disk to memory—Chapter 4.
3. Commands related to the control of the disks and the disk operating system are given in Chapter 5.
4. The printer can be operated directly from the keyboard to LIST and RUN programs, by using the following procedure. These are keyboard

Table 1-3. Specific Commands

Function	Built-in Before DOS	User Defined After DOS
Input Keyboard to memory	Implied IN#0	PRINT D$; "IN#0"
Tape to memory	Specialized ROM (LOAD)	Specialized ROM (LOAD)
Disk display-screen	None	MON I†
Disk to memory (SEQUENTIAL FILE)	None	PRINT D$; "OPEN FILE"* PRINT D$; "READ FILE" INPUT (RECORD)
Output Memory to CRT	Implied PR#0	PRINT D$; "PR#0"
Memory to tape	Specialized ROM (SAVE)	Specialized ROM (SAVE)
Memory to disk (SEQUENTIAL FILE)	None	PRINT D$; "OPEN FILE"* PRINT D$; "WRITE FILE" PRINT (RECORD)
Disk display-screen	None	MON O†
Memory to printer (printer card in slot #1. Instructions from within the program)		PRINT D$; "PR#1"—turns on printer card PRINT I$; "80N"—80 columns on printer PRINT Q$—select the printer (PRINT THE MATERIAL) PRINT S$—deselect the printer PRINT D$; "PR#0" —turn off printer (all control characters must be initialized before they are used)
Memory to printer (outside the program)		PR#1 —turn on printer card Control Q —select the printer LIST AND/OR RUN PROGRAM Control S —deselect the printer PR#0 —turn off printer card
Display disk commands on the screen		MON C†

*Files must always be closed at the end of a program.
From within the program — PRINT D$; "CLOSE FILE"
END
Outside the program — CLOSE—immediate execution command
†NOMON C, I, O turns monitor commands off

commands that are used with an INTEGRAL DATA SYSTEMS 440 printer, and the INPUT/OUTPUT printer interface card is in I/O slot No. 1.

a. Turn the printer interface card on; type PR#1 and RETURN *
b. To select the printer, press CTRL and Q simultaneously; RETURN and press the back arrow key once to prevent a syntax error.
c. When LIST or RUN is typed and RETURN, the program will list or run a hardcopy on the printer.
d. To deselect the printer, press CTRL and S simultaneously.
e. To turn the printer interface card off, type PR#0 RETURN.

The printer has dual in-line purpose (DIP) switches that control many functions, and should be set to output the correct results. The printer select, CTRL Q, and deselect, CTRL S, commands work in conjunction with the DIP switches.

5. Fig. 1-1 is an Applesoft program to create a sequential text file.

```
100 D$ =  CHR$ (4)
110  PRINT D$;"OPEN TEST FILE"
120  PRINT D$;"WRITE TEST FILE"
130  PRINT "WRITE TO DISK"
140  PRINT D$;"CLOSE TEST FILE"
150  END
```

Fig. 1-1. Program to open, write to disk, and close a sequential access file.

Although the program is rather short, it incorporates all necessary and proper techniques to create a file named "TEST FILE", and place the record "WRITE TO DISK" on the disk.

a. OPEN—the file on the disk.
b. WRITE—to this file on disk.
c. PRINT—this record on the disk.
d. CLOSE—the file.

6. Fig. 1-2 is an Applesoft program written to retrieve a sequential text file that is stored on disk.

```
100 D$ =  CHR$ (4)
110  PRINT D$;"OPEN TEST FILE"
120  PRINT D$;"READ TEST FILE"
130  INPUT A$
140  PRINT D$;"CLOSE TEST FILE"
150  END
```

Fig. 1-2. Program to open and read a sequential file—to input file records and close file.

a. OPEN—the file stored on disk.
b. READ—the file stored on disk.

*Keyboard commands will be in inverse type.

c. INPUT—the file on disk to memory.
d. CLOSE—the file.

In the creation of a file the WRITE and PRINT statements are used. In the retrieval of a file, the READ and INPUT statements are used.

Fig. 1-3 is a printout of the DOS COMMANDS WITHIN THE PROGRAM (Fig. 1-1) when MON I, O, C is used in the immediate execution mode—see Chapter 5.

```
MON I,O,C

RUN
OPEN TEST FILE
WRITE TEST FILE
WRITE TO DISK
CLOSE TEST FILE
```

Fig. 1-3. CRT display of MON I, O, C command resulting from program write to disk.

Fig. 1-4 is a printout of the DOS commands used to retrieve the sequential text file, which is generated by the program in Fig. 1-2, when MON I, O, C is used. The "?WRITE TO DISK" is the sequential text record that was stored on disk by the program in Fig. 1-1.

```
MON I,O,C

RUN
OPEN TEST FILE
READ TEST FILE
?WRITE TO DISK
CLOSE TEST FILE
```

Fig. 1-4. CRT display of MON I, O, C command resulting from program to read from disk and input to CRT.

Fig. 1-5 is an immediate execution printout of the sequential text record on disk. In Fig. 1-2, the record was read from disk, and INPUT A$ causes the record to be read into memory. Immediate execution of PRINT A$ causes the record WRITE TO DISK to be printed.

```
PRINT A$
WRITE TO DISK
```

Fig. 1-5. Sequential access record created by program write to disk.

Fig. 1-6 is a list of control codes used with keyboard, DOS, and the printer.

Table 1-4 is a list of the ASCII (American Standard Code for Information Interchange) character codes showing the decimal and hexadecimal number that represents the alpha, numeric, or special characters.

Table 1-5 shows the position of the seven bits of the binary representation of the ASCII code.

Table 1-4. ASCII Character Codes

Code		Char	Code		Char	Code		Char	Code		Char
Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
0	00	NUL	32	20	SP	64	40	@	96	60	
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	—	127	7F	DEL

Table 1-5. Bit Positions of Binary ASCII Code

Bit Numbers →								0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_7 ↓	b_6 ↓	b_5 ↓	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
			0	0	0	0	0	NUL	DLE	SP	0	@	p		p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	‖	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
			0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
			1	0	1	1	B	VT	ESC	+	;	K	[	k	{
			1	1	0	0	C	FF	FS	,	<	L	\	l	|
			1	1	0	1	D	CR	GS	−	=	M	]	m	}
			1	1	1	0	E	SO	RS	.	>	N	^	n	~
			1	1	1	1	F	SI	US	/	?	O	−	o	DEL

```
            CONTROL CHARACTERS

A$=CHR$(1)   THROW AWAY CHARACTER TO SOLVE DOS PROBLEM WHEN U
SING GET A$. CHR$(1) MAKES GET A$ DISPLAY THE 1ST CHARACTER
ON THE SCREEN WHEN PRINTING FILES. T$=CHR$(1): PRINT T$; A$
(I)

A$=CHR$(1)   CAUSES PRINTER TO PRINT IN THE ENHANCED MODE.
B$=CHR$(2)   CAUSES PRINTER TO PRINT IN THE NORMAL MODE.
C$=CHR$(3)   CAUSES PRINTER TO PRINT IN THE GRAPHICS MODE.
D$=CHR$(4)   CONTROLS THE DISK OPERATING SYSTEM (DOS).
I$=CHR$(9)   SETS PRINTER SPECIFICATIONS.
J$=CHR$(10)  CAUSES THE PRINTER TO LINEFEED.
K$=CHR$(11)  CAUSES THE PRINTER TO VERTICAL TAB.
L$=CHR$(12)  CAUSES THE PRINTER TO FORM FEED.
M$=CHR$(13)  RETURN KEY FROM THE KEYBOARD.
M$=CHR$(13)  CAUSES A CARRIAGE RETURN ON THE PRINTER.
Q$=CHR$(17)  SELECTS THE PRINTER AFTER THE PRINTER AND THE PR
INTE CARDS ARE TURNED ON.

S$=CHR$(19)  DESELECTS THE PRINTER.
   CHR$(27)  THE SAME EFFECT AS PRESSING THE ESCAPE KEY.
```

Fig. 1-6. Control characters used with keyboard, DOS, and printer.

Fig. 1-6 and Tables 1-4 and 1-5 are given to augment the rest of the library. In computer publications, references are continually given to the binary, decimal, and hexadecimal representation of the ASCII code. The reference to the ASCII code seldom gives a related figure or table to comprehend the relationship between the numbers and the characters. Fig. 1-6 and Tables 1-4 and 1-5 may not be completely related to the material, but they are useful in the overall relationship to disks, files, and printers.

With the information presented in this chapter, the user should understand the general relationship between disks, files, and printers, and be able to use them with limited success.

chapter 2

Disks

There are three basic magnetic storage units used in computer systems: drum units, tape units, and disk units. At the present time, tape and disk units are most frequently used. Most microcomputers use tape and a cassette player as a first type of mass storage. As the user becomes more sophisticated in the use of the computer, a disk operating system and printer are added.

DISK INFORMATION

A disk (Fig. 2-1) is a storage medium that combines bulk storage, fast access time, nonvolatility (the data is retained when the power is turned off), and relatively low cost to purchase and maintain.

The 5¼ inch floppy disk used with the Apple® computer has a storage capacity of 118K bytes in the 3.2 disk operating system (DOS 3.2), and 146K bytes in DOS 3.3. The DOS 3.3 disk will store between 100 and 120 pages of a normal sized book.

A disk (Fig. 2-1) is a round piece of magnetic tape. The disk is also called a diskette, flexible disk, floppy disk, or minidisk. The disk comes in two sizes, an 8 inch disk and a 5¼ inch disk.

The magnetic disk is made of mylar plastic and microscopic particles of ferric oxide in a liquid binder coat the mylar. The oxides are magnetically hard. When the oxides have taken a magnetic charge, they are arranged in a

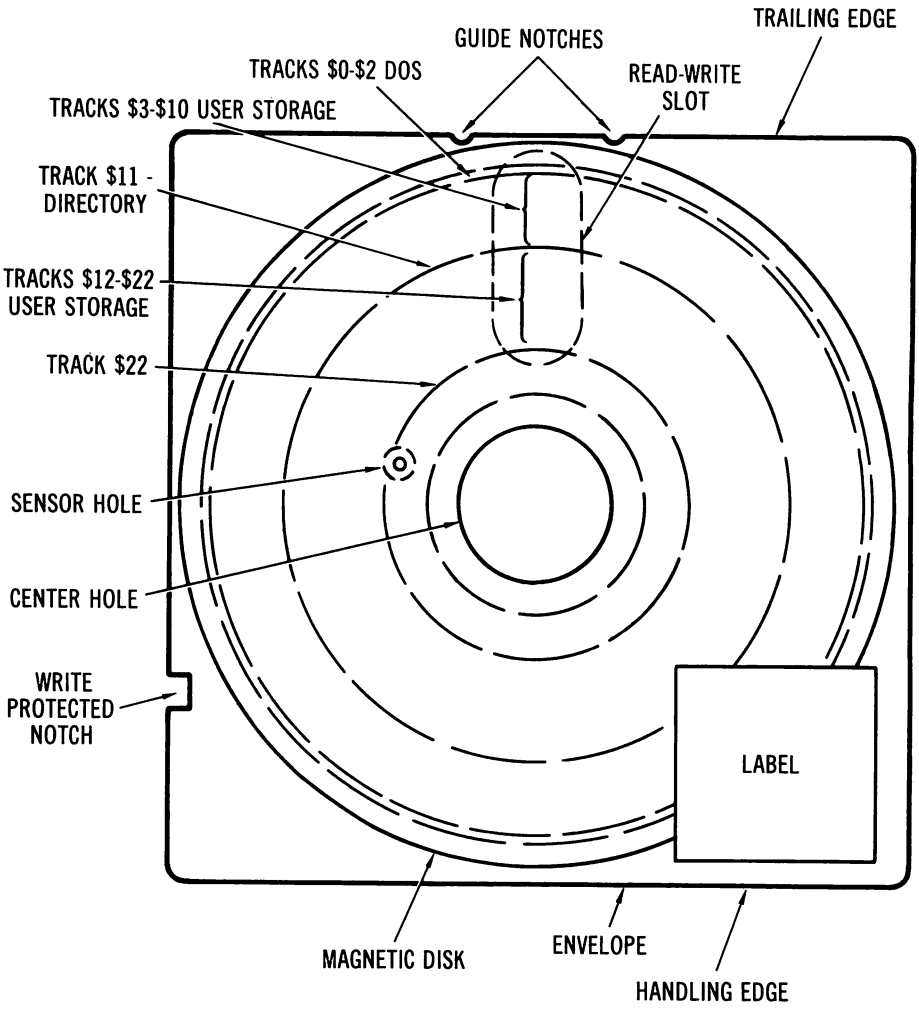


Fig. 2-1. Disk and envelope.

stationary, semipermanent manner. A strong magnet, such as a bulk demagnetizer, or a recording head, will rearrange the oxide particles.

The plastic covered container that covers the disk serves two purposes: (1) to protect the disk, and (2) to provide a stationary medium in which the disk can rotate.

The mylar disk is covered with a soft lubricant that reduces friction between the read-write head, and permits easy rotation within the envelope.

When in use, the disk rotates at 300 RPM. A slot in the envelope permits the read-write head access to the disk, to read and write data.

There are two holes through the envelope and the disk. The larger hole in the center of the disk is to accommodate the rotational drive to spin the disk at 300 RPM inside the envelope. The small hole is called the sector hole. The sector hole is a point of reference to which all recording takes place. When the sector hole passes under a sensor light, the drive acknowledges this to ensure a correct positioning to the disk. The recording space is divided into 35 tracks, and each track is divided into 16 sectors through DOS 3.3 software control. This is known as soft-sectoring of the disk.

The disk drive, or disk transport, has a movable read-write head. This head is controlled by a stepping motor that receives electrical pulses to control the movement of the head. If the pulse is high (1), the head is moved inward toward the next higher track number. If the pulse is low (0), the head is moved outward toward the next lower track number. When track $00 (hexadecimal zero) is reached, a signal indicates to the stepping motor to stop the head movement. The process of positioning the head on a particular track is called seeking.

When the disk is inserted in the drive, a motor drive spindle engages the magnetic disk through the center hole. When the proper DOS commands are received, the drive motor rotates the disk. The drive motor and disk envelope prevent the disk from turning at speeds greater than 300 RPM.

The read-write head contacts the surface of the disk when data is being transferred. A lubricant on the disk prevents excessive head and disk wear.

The life of the head is specified at 10,000 hours. Disk head contact for 24 hours per day would give a head life of approximately 1 year.

The disk head contact against the disk is specified at 40 hours before the error rate rises above unacceptable levels. Some manufacturers' flexible disks are guaranteed to read and write 100% error free for the period of 1 year after purchase.

While disks are reasonably rugged, they should not be bent, written on, or mishandled. The harshest object that should touch a disk is a cotton tip with slight pressure. Bending the disk can cause the magnetic particles to be rearranged and lose data. Disks should be protected from smoke, fingerprints, and dust particles. In a size relationship, Fig. 2-2, with the ferric oxide coating as one unit in size, a smoke particle is two units in size.

Preventative precautions should be taken to protect the valuable data stored on disks. Disks should be stored vertically in protective coverings in a cool place out of the sunlight.

FERRIC OXIDE FILM	1	UNIT
SMOKE PARTICLE	2	UNITS
FINGERPRINT	4	UNITS
DUST PARTICLE	10	UNITS
HUMAN HAIR	20	UNITS

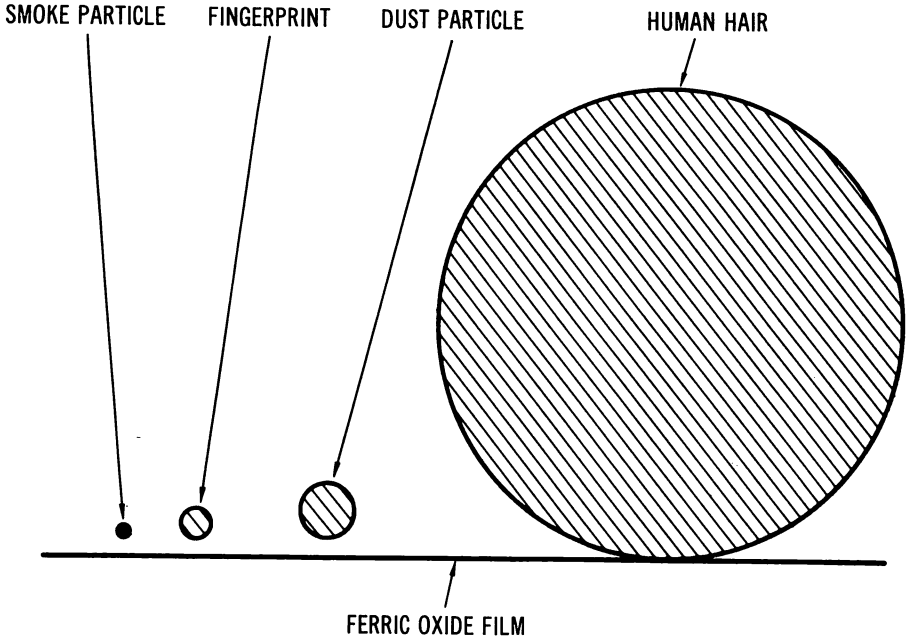


Fig. 2-2. Comparative sizes of disk debris.

DISK TRACKS AND SECTORS

A disk is divided into 35 tracks and each track is divided into 16 sectors. The 35 tracks are numbered from hex $00 starting on the outer edge, and are concentric rings increasing to hex $22 toward the center. Each track is broken into 16 sectors for DOS 3.3, and each sector contains 256 bytes of storage. Two sectors will hold one full screen page of a program.

The disk has a directory that holds all the information about programs, or files, stored on the disk. The disk operating system has a scheme for placing information on the disk.

When the disks are used in any manner, a request for use is made to the

disk operating system. This is a software program that is part of the DOS system and comes on the master disk. When the DOS system is activated, either by turning the computer on (if it has auto-start), or typing "IN#" or "PR#" command the DOS controller card reads the DOS system into memory and runs it. The DOS system reduces the computer's RAM memory by the amount required by the system. The DOS system will accept a range of commands to place data, information, programs, and other files on the disk. The DOS system does all the necessary checking to properly handle information going to disk. If the command given is incorrect the DOS system prints error messages on the screen.

The DOS takes approximately 10½K bytes of RAM memory for the system and the first three files and approximately ½K for each additional file.

The disk operating system is continually being updated. Newer, more flexible versions that create more storage space on the same disk area are produced, and offered to the public. The current version is DOS 3.3, and has just been introduced. Since these changes will probably continue, the information is continuously being outdated. The conversion from one DOS version to another is relatively simple. The conversion can be made by following the instructions in the manual.

chapter 3

Using the System

The disk operating system consists of a disk drive, or disk drives, an interface card that plugs into one of seven INPUT/OUTPUT slots (zero slot is reserved for the language card), and a master disk. When the disk operating system is properly connected to the computer, the master disk is placed in the disk drive that will activate the system. The monitor is turned on. The disk drive door is closed, and the Apple OFF-ON switch is placed in the ON position.

If the red light on the front panel of the disk drive is operating properly, the light shines when the read-write head is in use.

APPLE II is displayed on the upper part of the CRT. The prompt of the appropriate language is viewed to the left of the flashing cursor. DOS can be activated either in Integer BASIC or Applesoft.

DOS MASTER

The DOS system master is a very special disk. The DOS is on this disk, as well as many valuable programs that can aid the programmer. The RENUMBER program is especially useful in building a large program. The beginning programmer can learn many details of correct programming techniques from studying these programs.

BACKUP DISKS

Many sources recommend that one or more backup copies of all masters be made in case of loss or destruction. This duplication is fine to a point. When there are many master disks, a duplicate of each copy soon produces a glut of duplicate, dust gathering disks. Judicious duplication of masters can save space and money. Careful handling of disks can be a form of duplication.

When the system is operating, disks must be initialized so programs and files can be stored for use by the system. This initialization of disks transfers the disk operating system from one disk to another, so DOS can be activated by any initialized disk. INIT is the basic word that sets up the tracks and sectors and transfers DOS to a new disk.

INITIALIZE A DISK

To initialize a disk for the first time, follow this procedure. The DOS has been activated by reading the master disk.

1. The master disk is removed from disk drive No. 1. On a multidrive system, drive No. 1 is referred to as the boot drive.
2. The disk to be initialized is placed in the boot drive.
3. Memory is cleared by typing NEW and RETURN.
4. An informational program is typed to acknowledge the transfer of the DOS, the size of the computer memory on which the system was initialized, and the date of initialization.

```
10 PRINT "DUPLICATE DOS SYSTEM CREATED ON A 48K SYSTEM"
20 PRINT "15- NOV- 1980"
30 END
```

5. When the program has all the information necessary for the user's needs, the disk can be initialized. Other useful information can be transferred to the disk at the time of initialization. The disk can be given a volume number, and the slot and drive options can be used to reference the disk drive.
   a. V10—disk volume #10.
   b. S6—I/O slot #6.
   c. D2—disk drive #2.

   Any, or all, of these options may be transferred during initialization. At least, each disk should be initialized with a volume number, so initialized disks can be properly sequenced.
6. INIT DUPLICATE, V1 is typed on the screen RETURN. The disk will

spin for about 2 minutes until the initialization process is complete. When the initialization is complete, the language prompt and the cursor return to the CRT. The red disk drive light also goes dark.

Sometimes when the disk drives operate, they make a very disturbing noise. It sounds as if the mechanism is out of balance and the grinding sound is similar to fingernails scratching across a blackboard. The drives seem to be tearing apart. The sound may be normal for the unit, but is very disquieting. An authorized head cleaning kit and fluorocarbon isopropyl alcohol, or pure denatured alcohol, can help reduce the grinding sound. Apple dealers say the sound is normal and seem to think nothing of it. The author's drive units have never given any trouble, and the sound level was greatly reduced after cleaning.

The first disk has now been initialized. If other disks are to be initialized, initialize them with a volume number so they can be properly sequenced.

ACCESSING THE DISK DRIVES

In a computer system with more than one drive, it is important to remember how to access the proper disk. The boot drive in a two disk drive system is assigned ",D1", and the other drive is assigned ",D2". The most reliable way to access the desired disk is to add the suffix, D1 or D2, to the command—LOAD PRIME, D1, or LOAD PRIME, D2.

Another way to access the proper disk is to remember which disk was last used. If a drive is not specified, the last disk accessed will be accessed on the next command—this is known as the default drive. The default drive changes each time the other disk is accessed.

The default drive is the last drive accessed. If the system was just booted, the boot drive is the default drive. If no drive was specified, the last drive used will be accessed again.

DISKS THAT WILL NOT INITIALIZE

Infrequently, a new disk will not initialize. The new disk is placed in the drive, and all procedures are correctly executed to initialize the disk. Sometimes the initialization routine produces an I/O error that is printed on the screen. Sometimes the initialization routine causes the disk to rotate endlessly, and produces no error message. When the correct initialization routine fails to initialize the disk, check the disk envelope (Fig. 3-1).

On the leading edge of the envelope (opposite the edge handled to place

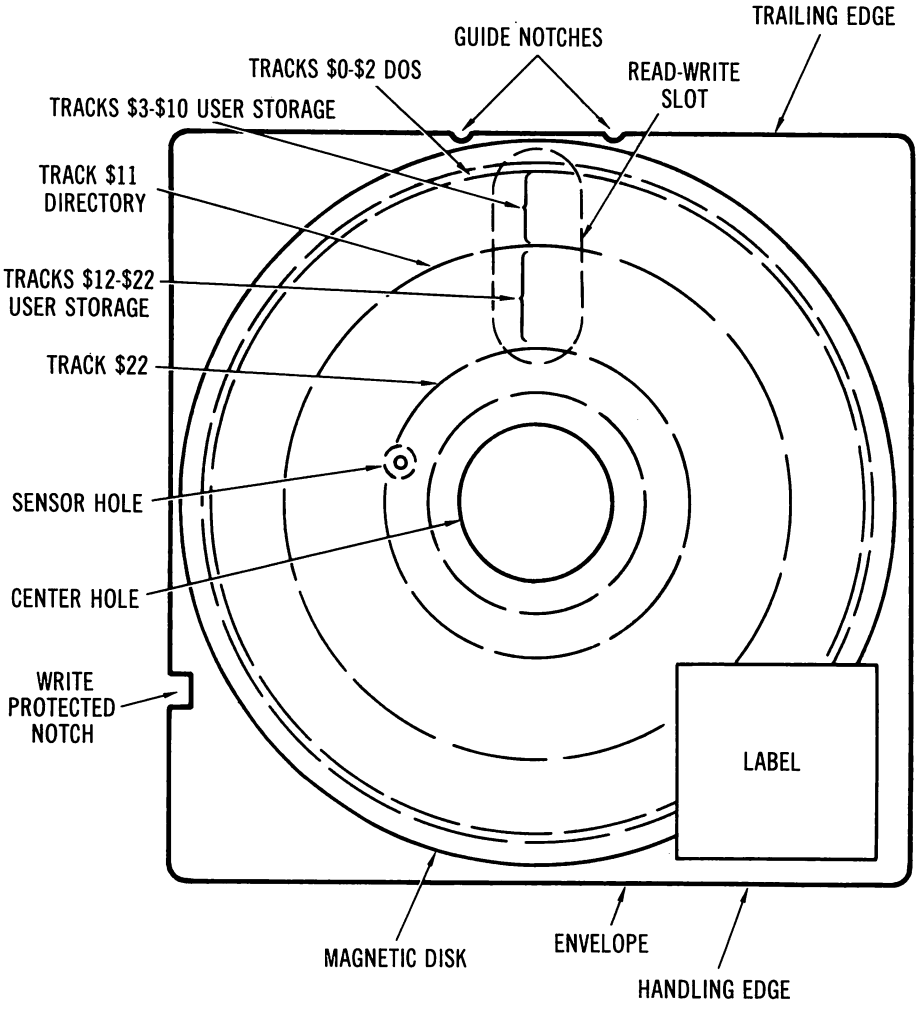


Fig. 3-1. Disk and envelope.

the disk in the drive) are two half circle guide notches that seat against the forward ridges on the inside of the disk drive. At times these notches will not be completely cut through the envelope. The solution to the problem is to cut the envelope notches so the disk will seat properly in the drive. A surgically sharp modeling knife, or utility knife, can be used to cut cleanly through the plastic coated envelope from top to bottom within the limits of the established notches. Do not cut past the established notch boundary. The magnetic disk is about three millimeters away from the outer limits of the

envelope. Do not cut the magnetic disk. When the guide notches are "cleaned out," the disk should initialize.

Another reason a disk will not boot, or catalog, is that it will not rotate within the envelope. This can be caused by the envelope being slightly crushed. It could have been crushed due to rough handling during manufacture, packing, or shipping.

To try to recover the data on the disk, take the disk out of the crushed package, and place the disk in a vertical position in an unrestricted area. This takes the pressure off the disk. The plastic coated envelope is said to have a "memory," and attempts to return to its original manufactured shape. After a few days in a vertical position, place the disk in the disk drive and leave the disk drive door open. Type CATALOG RETURN, and when the disk drive starts to rotate, close the door on the drive. The disk drive rotation might cause the disk to rotate in the envelope.

If this does not cause the disk to rotate in the envelope, grasp the envelope between the thumb and first finger. Place the first and second fingers of the other hand in the center hole and spread the fingers to put pressure on the disk. Try to rotate the disk within the envelope, by turning the envelope clockwise, and the disk counterclockwise.

If the disk can be rotated in the envelope try to recover the data on disk by loading it to memory, and then saving the data to another disk, or to tape. It is suggested the crushed disk not be used again after the data has been recovered.

USING THE VERIFY COMMAND

The VERIFY command can be used to check the sectors of the disk on which programs are stored. The VERIFY command is explained in detail in Chapter 4.

chapter 4

Loading and Saving Programs on Disks

Using disks for storage, rather than cassette tape, makes computing much quicker and easier. The boring, time consuming routine of loading and saving programs on tape has mostly been eliminated. With disk drives the computer seems to be a completely different machine.

After the DOS system is activated and memory has been cleared by typing NEW, you are ready to type a program on the keyboard, and save it to disk.

Type in the following program which computes and prints the prime numbers in an integer.

```
100 PRINT "INPUT A NUMBER ";:INPUT N
110 F = 0
120 PRINT N;
130 D = 2
140 IF N/D = INT (N/D) THEN 190
150 D = D + 1
160 IF D <= SQR(N) THEN 140
170 IF F = 1 THEN PRINT "*";N : GOTO 999
180 PRINT "IS PRIME" : GOTO 999
190 IF F = 0 THEN PRINT "=" ; : F = 1 : GOTO 210
200 PRINT "*" ;
210 PRINT D;
```

```
220 N = N/D
230 GOTO 130
999 END
```

RUN the program to determine that it works and that there are no BUGS in it. For the present time name the program PRIME.

SAVE A PROGRAM TO DISK

To SAVE the program named PRIME on disk, type

SAVE PRIME RETURN.

The red light on the default drive turns on, the disk rotates, and the program is SAVEd under the name PRIME.

SAVE and SAVE PRIME are two different commands. SAVE is a command that saves a program to cassette tape. SAVE will not cause DOS to be activated. The DOS command that saves a program to disk is SAVE (program name)—SAVE PRIME.

CATALOG A DISK

To determine that a program has been saved, type CATALOG. CATALOG is the DOS command to display the names of the programs stored on disk on the CRT. Type

CATALOG RETURN

The disk rotates, the programs on the screen scroll downward, and the catalog on the disk is displayed on the CRT. Since the program was typed using Applesoft, the "A" to the left of the program PRIME designates the Applesoft language. The catalog can display four different types of files.

A—Applesoft
B—Machine or Binary
I—Integer BASIC
T—Text

The catalog display shows the following configuration:

```
A 002 DUPLICATE
A 002 PRIME
```

The 002 represents the number of sectors used by the program. Each disk sector contains 256 bytes, and each program fills two sectors on the disk.

Now that PRIME has been saved to disk, and the catalog read, the program is now loaded into memory.

LOAD A PROGRAM FROM DISK

LOAD and LOAD PRIME are two different commands. LOAD is reserved for loading tape into memory, while LOAD PRIME is a DOS command to load a program from disk to memory. Type

NEW RETURN

to clear memory. Type

LOAD PRIME RETURN

Again the red light on the disk drive shines, and the disk rotates. When the prompt and the cursor return to the CRT, the program is loaded.

To list the program on the screen, type LIST (no program name) and RETURN.

RUN A PROGRAM FROM DISK

Another way in which a program can be loaded from disk to memory is to type RUN (program name)—RUN PRIME and RETURN. This command loads and runs the program named PRIME.

There are two ways to load a program to memory from disk.

1. LOAD (program name)—LOAD PRIME and RETURN.
2. RUN (program name)—RUN PRIME and RETURN.

Now that the euphoria of successfully saving a program to disk, and loading a program from disk, has subsided, some questions need to be answered.

PROGRAM NAMES

What is the maximum length of a program name?

A program name can contain up to 30 alphanumeric, and/or special characters, and control characters, except a comma. The comma is used to end the program or file name. The information following the comma is the slot or drive option. The program name cannot start with a numeric or special character.

Examples of legal names are as follows:

1. PRIME
2. PRIME-1
3. PRIME*().PRIME

Examples of illegal names are as follows:

1. A name that contains more than thirty characters, including spaces.
2. 7 COME 11
3. , PRIME

To load a program from disk, the program name must be typed exactly as the program name on the disk catalog. Any deviation from the catalog name will produce a FILE NOT FOUND on the CRT. To prevent excess typing, keep the program name short, and make it pertinent to the subject in the program.

ANOTHER WAY TO LOAD PROGRAMS

When LOADing programs, there is a satisfactory method to prevent excess typing. Type CATALOG to display the programs on the CRT, and enter the edit mode by pressing the ESC key.

1. Type CATALOG RETURN.
2. Type LOAD.
3. In Applesoft press ESC to enter the edit mode.
4. Using the pure cursor moves mode keys (I, J, K, M) move the cursor to the blank space between the sector count number and the program name.

```
002■ PRIME
```

5. Press the SPACE BAR to disengage the pure cursor moves mode.
6. Use the right arrow key and repeat key to move the cursor past the last character in the program name.

```
002 PRIME■ RETURN
```

The program loads. Using this method to load programs, there is minimal typing and little chance of typing an incorrect program name.

CONTROL CHARACTERS

Control characters (Fig. 1-6) are not printed on the screen. If a control character is placed in a program name, the program will not load unless the control character is typed exactly as in the original program name.

A control character may be typed accidentally into a program name, or a control character can be a secret key to control the loading of a program.

APPLE II, THE DOS MANUAL, published by Apple Computer, Inc., shows a program written especially to detect control characters that have been placed in a program. The program is shown on page 151 of their manual. This program detects all control characters except Control M—RETURN, ESC, CTRL H —left arrow, and CTRL U —right arrow.

WHICH DISK DRIVE IS ACCESSED?

Another question to be answered, how is more than one disk drive accessed? In Chapter 3, the disk interface card was placed in I/O slot No. 6, and the disk volume was assigned the number 10.

When a program is to be saved to disk drive No. 1, type

```
SAVE PRIME,D1 RETURN
```

The ",D1" sends the saved program to disk drive No. 1. When a program is to be saved to disk drive No. 2, type

```
SAVE PRIME,D2 RETURN
```

The same procedure is used with RUN PRIME,D2.

Does the command, SAVE PRIME, V10, select the proper disk volume, without the ",D1", or ",D2"? If volume No. 10 is in the default drive the SAVE PRIME is successful. If volume No. 10 is not in the default drive, an error message will be printed on the CRT. The volume selector does not cause the system to search the other disk drives to check to see if volume No. 10 is in another disk drive.

DELETING PROGRAMS FROM DISK

Programs can be DELETEd from the disk by using the command, DELETE (program name)—DELETE PRIME. If the program name is incorrectly spelled, the wrong program may be deleted. The machine is not concerned with what you intend to do. It is only concerned with what you do. (DELETE is a DOS command, and DEL is an EDIT command.)

One method to overcome the possibility of incorrect deletion is to use the following procedure:

1. Type CATALOG RETURN.
2. Type DELETE.
3. Press ESC to use the pure cursor moves mode.

4. Move the cursor to the blank between the sector count and the program name using I, J, K, or , M.

   002 ■ PRIME

5. Press the space bar one time.
6. Press forward arrow and repeat until the cursor is beyond the last letter of the program name.

   002 PRIME ■ RETURN

   Using this method you can delete the correct program without typing the program name.
7. Control X will stop this procedure anytime before RETURN is pressed.

RENAMING PROGRAMS

Programs can be renamed by using the command RENAME in the following format:

RENAME PRIME, PRIME-1 RETURN

There are two companion commands, LOCK, and UNLOCK. LOCK (program name, volume number, slot number, or disk drive number)—LOCK PRIME prevents the program from accidentally being erased. An asterisk is placed to the left of the program that is locked.

* A 002 PRIME

A locked program can be loaded and run in the normal way. The lock prevents the program from being DELETED or RENAMED. If you attempt to DELETE or RENAME a locked program A FILE LOCKED message will be placed on the screen.

The companion command to LOCK is UNLOCK. UNLOCK (program name, volume number, slot number, or disk drive number)—UNLOCK PRIME takes away the asterisk and the program can be deleted or renamed.

VERIFY COMMAND

VERIFY is a command used to check the condition of a disk. The disk could have been slightly crushed, dirty, or have imperfections on the recording surface. VERIFY (program name, volume number, slot number, or disk drive number)—VERIFY PRIME and a program on the disk that is properly recorded on these sectors will return the correct language prompt and the cursor to the screen.

> for Integer BASIC
] for Applesoft
* for Monitor

If VERIFY PRIME finds imperfections on the recording surface it returns I/O ERROR.

VERIFY does not give assurance that PRIME is free of bugs and will run correctly. VERIFY will not correct a faulty program.

WRITE PROTECT FEATURE

If all programs on a disk are to be protected, use the "write protect" feature. The disk envelope has a write protected notch, Fig. 3-1. To write protect a disk, cover this notch with one of the write protect seals that come with the disks. This seal is placed over the notch so the disk cannot be recorded on, or erased.

Some MASTERS come write protected. The envelope has no write protect cut out notch. The programs on most write protect disks can be copied to make duplicate disks.

The FORT 2 MASTER, for Apple FORTRAN, is write protected, and the compile program cannot be copied.

To protect the disk when pressing RESET , or turning off the system, open the door, or doors, on the disk drives. When the door on the disk drive is opened, the read-write head does not engage the disk. Pressing RESET or turning off the system causes an electrical pulse to jerk the read-write head, and possibly damage the disk.

chapter 5

DOS Commands

The Apple computer has several ways to load a language into memory. The original Apple had Integer BASIC in ROM as a primary language, and Applesoft was loaded from tape. There is a language card that has Applesoft in ROM, and plugs into I/O slot No. 0. The Apple now has two languages in ROM. The Apple II Plus has Applesoft in ROM and Integer BASIC is provided on a language card. The Pascal language card plugs into I/O slot No. 0, has RAM memory that can load Integer BASIC, Applesoft, Pascal, or FORTRAN. The DOS 3.3 loads Integer BASIC or Applesoft from disk into the language card RAM memory. The DOS stores the language that is not stored in ROM.

CHANGE LANGUAGES

In an Apple computer that has both Integer BASIC and Applesoft, it is easy to transfer from one language to the other.

To transfer from Integer BASIC to Applesoft type FP (for floating point) RETURN .

To transfer from Applesoft to Integer BASIC type INT RETURN .

A procedure to reset pointers to the memory empty condition when transferring from one language to another is to type NEW after the transfer.

1. FP.....INT
2. INT.....FP
3. NEW.....NEW

Transferring from one language to another causes the program in memory to be lost. Before transferring from one language to another, with a program in memory, SAVE THE PROGRAM IN MEMORY.

INIT, SAVE, LOAD AND CATALOG

DOS commands, INIT, SAVE, LOAD, and CATALOG, used in the immediate execution mode are discussed in Chapter 4. How can these commands be used in the deferred execution mode, that is, while the program is running?

CONTROL D IS THE KEY TO DOS

Control D is the key to DOS. Any deferred execution DOS statement must be preceded by Control D. The DOS looks at the output from every print statement in a program, but only acts on those that are preceded by Control D.

There are reserved words that activate DOS when used in the immediate execution mode. Three of these reserved words are CATALOG, SAVE, and LOCK. To demonstrate that DOS is not activated in a program without Control D, the Applesoft program in Fig. 5-1 was written. The program contains three reserved DOS words, CATALOG, SAVE, and LOCK. When the program is RUN, Fig. 5-2, the reserved words and parameters, CATALOG, SAVE DUMMY, LOCK DUMMY, and CATALOG, are printed, but the DOS system is not activated.

```
100  REM :PROGRAM DEMONSTRATING  COMMANDS WITHOUT 'PRI
     NT D$;'
110 D$ =  CHR$ (4)
120  GOSUB 210
130  PRINT "CATALOG"
140  GOSUB 210
150  PRINT "SAVE DUMMY"
160  GOSUB 210
170  PRINT "LOCK DUMMY"
180  GOSUB 210
190  PRINT "CATALOG"
200  END
210  RETURN
```

Fig. 5-1. Applesoft program without PRINT D$;.

```
 RUN
CATALOG
SAVE DUMMY
LOCK DUMMY
CATALOG

]PR#0
```

Fig. 5-2. RUN of Applesoft program without PRINT D$;—words print on the CRT but do not activate DOS.

PRINT D$

Fig. 5-3 is an Applesoft program similar to the program in Fig. 5-1, but line 210—PRINT D$ has been added. Each GOSUB 210 caused Control D to be printed before the reserved DOS word. Fig. 5-4, the RUN of Fig. 5-3, demonstrates when Control D preceded the reserved DOS word, DOS is activated. Fig. 5-4 shows the catalog stored on the disk. The program DUMMY is saved on disk, DUMMY is locked, and the program causes a second CATALOG of the disk to be printed. The second CATALOG shows the file DUMMY was created, saved, and locked, when Control D preceded the reserved DOS words.

The programs in Figs. 5-1 and 5-3 are similar except Control D is printed before the reserved DOS word in Fig. 5-3.

MON C, I, O

MONITOR COMMAND, INPUT, and OUTPUT (MON C, I, O) is a multiple DOS command that causes the commands to DOS, input to DOS, and output from DOS to be printed on the CRT. The Applesoft program in Fig. 5-5 was written to demonstrate how Control D, and MON C are used to print DOS commands on the CRT. The program in Fig. 5-5 is similar to the program in Fig. 5-3. One difference is line 115—PRINT D$; "MON C",

```
100  REM :PROGRAM DEMONSTRATING  COMMANDS WITH 'PRINT
     D$;'
110 D$ =  CHR$ (4)
120  GOSUB 210
130  PRINT "CATALOG"
140  GOSUB 210
150  PRINT "SAVE DUMMY"
160  GOSUB 210
170  PRINT "LOCK DUMMY"
180  GOSUB 210
190  PRINT "CATALOG"
200  END
210  PRINT D$;
220  RETURN
```

Fig. 5-3. Applesoft program with PRINT D$; and DOS commands.

```
 RUN

DISK VOLUME 254

 A 002 DISKINIT
 A 002 FILE STARTER
 A 002 FILE READER
 I 002 FILE STARTER-IB
 I 002 FILE READER-IB
 A 002 CH 5 COMMANDS W/O D$
 A 002 CH 5 COMMANDS WITH D$
 A 002 CH 5 DOS COMMANDS - MON C
 I 003 CH 5 COMMANDS W/O D$-IB
 I 003 CH 5 COMMANDS WITH D$-IB
 I 003 CH 5 DOS COMMANDS - MON C-IB


DISK VOLUME 254

 A 002 DISKINIT
 A 002 FILE STARTER
 A 002 FILE READER
 I 002 FILE STARTER-IB
 I 002 FILE READER-IB
 A 002 CH 5 COMMANDS W/O D$
 A 002 CH 5 COMMANDS WITH D$
 A 002 CH 5 DOS COMMANDS - MON C
 I 003 CH 5 COMMANDS W/O D$-IB
 I 003 CH 5 COMMANDS WITH D$-IB
 I 003 CH 5 DOS COMMANDS - MON C-IB
*A 002 DUMMY

]PR#0
```

Fig. 5-4. RUN of Applesoft program with PRINT D$; and DOS commands.

which turns on the monitor command to disk, and prints the commands on the CRT. Another difference is line 195—PRINT D$; "NOMON C", which turns off the monitor command.

Fig. 5-6 is a RUN of the program in Fig. 5-5. Because the PRINT D$; MON C is present in the program in Fig. 5-5, the DOS command, CATALOG is printed, the disk catalog is printed, SAVE DUMMY, and LOCK DUMMY are printed, and the second disk catalog is printed.

Figs. 5-9 through 5-13 are Integer BASIC programs and RUNS and are similar to the Applesoft programs and RUNS in Figs. 5-1 through 5-6.

```
100  REM :PROGRAM DEMONSTRATING  COMMANDS WITH MONITOR

110 D$ =  CHR$ (4)
115  PRINT D$;"MON C"
120  GOSUB 210
130  PRINT "CATALOG"
140  GOSUB 210
150  PRINT "SAVE DUMMY"
160  GOSUB 210
170  PRINT "LOCK DUMMY"
180  GOSUB 210
190  PRINT "CATALOG"
195  PRINT D$;"NOMON C"
200  END
210  PRINT D$;
220  RETURN
```

Fig. 5-5. Applesoft program with PRINT D$; and MON C.

IMMEDIATE AND DEFERRED COMMANDS

There are two general classes of DOS statements and commands: (1) deferred execution statements that are used to create and handle text files while a program is running, and (2) statements or commands that may be used either in the deferred or immediate execution mode.

The DOS statements that should be used in the deferred execution mode from within a program are used to create and handle text files.

```
PRINT D$; "OPEN TEXTFILE"—OPEN
```

OPEN is used to open a file on disk. If the file does not exist, then it will be created. If it already exists, then its contents will be made accessible to the user.

```
PRINT D$; "WRITE TEXT FILE"—WRITE
PRINT
```

Use the print which is provided with the language. (WRITE and PRINT are used together)

```
PRINT D$; "READ TEST FILE"—READ
INPUT A$
```

INPUT is a command that is provided by the language you are working in. (READ and INPUT are used together)

```
PRINT D$; "CLOSE TEST FILE"
PRINT D$; "APPEND TEST FILE"
```

```
 RUN

CATALOG

DISK VOLUME 254

 A 002 DISKINIT
*A 002 FILE STARTER
*A 002 FILE READER
*I 002 FILE STARTER-IB
*I 002 FILE READER-IB
*A 002 CH 5 COMMANDS W/O D$
*A 002 CH 5 COMMANDS WITH D$
*A 002 CH 5 DOS COMMANDS - MON C
*I 003 CH 5 COMMANDS W/O D$-IB
*I 003 CH 5 COMMANDS WITH D$-IB
*I 003 CH 5 DOS COMMANDS - MON C-IB
*A 004 CH 8 CREATE SEQUENTIAL
*A 011 CH 8 UPDATE SEQUENTIAL
*A 008 CH 8 READ SEQUENTIAL

SAVE DUMMY
LOCK DUMMY
CATALOG

DISK VOLUME 254

 A 002 DISKINIT
*A 002 FILE STARTER
*A 002 FILE READER
*I 002 FILE STARTER-IB
*I 002 FILE READER-IB
*A 002 CH 5 COMMANDS W/O D$
*A 002 CH 5 COMMANDS WITH D$
*A 002 CH 5 DOS COMMANDS - MON C
*I 003 CH 5 COMMANDS W/O D$-IB
*I 003 CH 5 COMMANDS WITH D$-IB
*I 003 CH 5 DOS COMMANDS - MON C-IB
*A 004 CH 8 CREATE SEQUENTIAL
*A 011 CH 8 UPDATE SEQUENTIAL
*A 008 CH 8 READ SEQUENTIAL
*A 002 DUMMY

NOMON C
```

Fig. 5-6. RUN of Applesoft program with PRINT D$; and MON C.

APPEND has no value in a random access (direct) file, and must be used in a special procedure in a sequential access file.

```
PRINT D$ "POSITION TEST FILE"
```

POSITION requires that the exact byte in the file be determined before position is used.

The following list of commands can be used either in the deferred or immediate execution mode.

BLOAD	BRUN	BSAVE
CATALOG	CHAIN	CLOSE
DELETE	IN #	EXEC
LOAD	LOCK	MON C,I,O
NOMON C,I,O	PR #	RENAME
RUN	SAVE	UNLOCK

Control D is the key to DOS commands. Any printout must be preceded by Control D to be recognized and acted on by the DOS.

CONTROL D IN APPLESOFT

In Applesoft, Control D can be created with CHR$(4), and Control D is initialized as D$ = CHR$(4). When Control D has been initialized in CHR$(4), then PRINT D$; can be used to activate Control D in DOS deferred execution mode statements.

CONTROL D IN INTEGER BASIC

In Integer BASIC, Control D must be initialized as a control character, stored in a string, that does not print on the screen. The format to initialize Control D in Integer BASIC is as follows.

1. Type the program line number—10, for example.
2. Type D$ = ".
3. Press CTRL and D.
4. Type the closing quotation mark—".

The initialization is displayed on the screen as

```
10 D$ = ""
```

and Control D is not visible on the CRT. Caution: if the cursor is passed over the statement in line 10, Control D will be dropped, and the DOS will not be activated using this initialization.

PROGRAM TO INITIALIZE CONTROL D

To overcome the handicap of not being able to determine if Control D has actually been initialized, a program is presented to be able to initialize

```
 10 REM  :CREATE TEST FILE
 20 DIM CHR$(27)
 30 FOR J=1 TO 26: POKE 2054+J,J+128: NEXT J
 40 POKE 2081,30
100 DIM A$(250)
110 D$=CHR$(4,4)
120 PRINT D$;"OPEN TEST FILE"
130 PRINT D$;"WRITE TEST FILE"
140 PRINT "WRITE TO DISK"
150 PRINT D$;"CLOSE TEST FILE"
160 END
```

```
LINE 30 POKE INTO MEMORY LOCATION 2054+J THE CONTROL CHARACT
ERS.

        J+128
        129=CONTROL A
        130=CONTROL B
        131=CONTROL C
        132=CONTROL D
         .
         .
         .
        154=CONTROL Z
         POKE 2081,30---30 IS AN END OF STRING CHARACTER
        DIM A$(250)-SETS INTEGER BASIC SO A$ CAN HOLD 250 CH
        ARACTERS
```

Fig. 5-7. Integer BASIC special statements to allow Integer BASIC to initialize D$ = CHR$(4) instead of D$ = "", and explanation of the program.

```
 RUN
WRITE TO DISK

> PR#0
```

```
 10 REM  :CREATE TEST FILE
 20 DIM CHR$(27)
 30 FOR J=1 TO 26: POKE 2054+J,J+128: NEXT J
 40 POKE 2081,30
100 DIM A$(250)
110 D$=CHR$(4,4)
120 PRINT D$;"OPEN TEST FILE"
130 PRINT D$;"READ TEST FILE"
140 INPUT A$
150 PRINT A$
160 END
```

Fig. 5-8. Integer BASIC program to retrieve the record created by the program in Fig. 5-7.

```
 10 REM :CREATE CHR$ VALUES
 20 DIM CHR$(27)
 30 FOR J=1 TO 26: POKE 2054+J,J+128: NEXT J
 40 POKE 2081,30
100 REM :PROGRAM DEMONSTRATING       COMMANDS WITHOUT 'PRI
    NT D$;'
110 D$=CHR$(4,4)
120 GOSUB 210
130 PRINT "CATALOG"
140 GOSUB 210
150 PRINT "SAVE DUMMY"
160 GOSUB 210
170 PRINT "LOCK DUMMY"
180 GOSUB 210
190 PRINT "CATALOG"
200 END
210 RETURN
```

Fig. 5-9. Integer BASIC program without PRINT D$;.

```
 RUN
CATALOG
SAVE DUMMY
LOCK DUMMY
CATALOG

>PR#0
```

Fig. 5-10. RUN of Integer BASIC program without PRINT D$;—words print on CRT but do not activate DOS.

```
 10 REM :CREATE CHR$ VALUES
 20 DIM CHR$(27)
 30 FOR J=1 TO 26: POKE 2054+J,J+128: NEXT J
 40 POKE 2081,30
100 REM :PROGRAM DEMONSTRATING       COMMANDS WITH 'PRINT
    D$;'
110 D$=CHR$(4,4)
120 GOSUB 210
130 PRINT "CATALOG"
140 GOSUB 210
150 PRINT "SAVE DUMMY"
160 GOSUB 210
170 PRINT "LOCK DUMMY"
180 GOSUB 210
190 PRINT "CATALOG"
200 END
210 PRINT D$;
220 RETURN
```

Fig. 5-11. Integer BASIC program with PRINT D$; and DOS commands.

```
>RUN

DISK VOLUME 254

 A 002 DISKINIT
 A 002 FILE STARTER
 A 002 FILE READER
 I 002 FILE STARTER-IB
 I 002 FILE READER-IB
 A 002 CH 5 COMMANDS W/O D$
 A 002 CH 5 COMMANDS WITH D$
 A 002 CH 5 DOS COMMANDS - MON C
 I 003 CH 5 COMMANDS W/O D$-IB
 I 003 CH 5 COMMANDS WITH D$-IB
 I 003 CH 5 DOS COMMANDS - MON C-IB
 A 004 CH 8 CREATE SEQUENTIAL
 A 011 CH 8 UPDATE SEQUENTIAL
 A 008 CH 8 READ SEQUENTIAL
 T 002 TEST FILE


DISK VOLUME 254

 A 002 DISKINIT
 A 002 FILE STARTER
 A 002 FILE READER
 I 002 FILE STARTER-IB
 I 002 FILE READER-IB
 A 002 CH 5 COMMANDS W/O D$
 A 002 CH 5 COMMANDS WITH D$
 A 002 CH 5 DOS COMMANDS - MON C
 I 003 CH 5 COMMANDS W/O D$-IB
 I 003 CH 5 COMMANDS WITH D$-IB
 I 003 CH 5 DOS COMMANDS - MON C-IB
 A 004 CH 8 CREATE SEQUENTIAL
 A 011 CH 8 UPDATE SEQUENTIAL
 A 008 CH 8 READ SEQUENTIAL
 T 002 TEST FILE
*I 003 DUMMY

>PR#0
```

Fig. 5-12. RUN of Integer BASIC program with PRINT D$; and DOS commands.

Control D in the same manner in Integer BASIC and Applesoft. For Integer BASIC programs add the following routine:

```
20 DIM CHR$(27)
```

```
30 FOR J = 1 TO 26 : POKE 2054 + J, J + 128 : NEXT J
40 POKE 2081, 30
```

This routine is used in the programs in Figs. 5-7, 5-8, 5-9, 5-11, and 5-13.

Before Integer BASIC programs are run using the preceding routine, low memory must be set by the immediate execution command—LOMEM : 2048. LOMEM : 2048 is the default value on all Apple computers.

Line 20 dimensions the CHR$ to reserve 27 memory locations to hold the 26 characters of the alphabet, and the number 30, which is an end of string character.

The routine at line 30 places 26 control characters in memory locations from 2054 (decimal) to 2080 (decimal).

Line 40 POKES the value of 30 into the memory location at 2081—30 is an end of string character.

To test your understanding of the DOS commands use the immediate execution command to DELETE the file DUMMY, that was created by the program in Fig. 5-5. Why can't DUMMY be immediately deleted? Now write a program to DELETE the LOCKed file DUMMY.

MONITOR COMMANDS

The monitor can be turned on by either the immediate or deferred execution mode command MON C, I, O. The monitor can be turned off by either the immediate or deferred execution mode command NOMON C, I, O.

Figs. 5-5 and 5-6 demonstrate the deferred execution mode of MON C, and NOMON C, used with the Applesoft language.

Figs. 5-13 and 5-14 demonstrate the use of deferred execution commands MON C, and NOMON C, used with the Integer BASIC language.

The MON C, used in Figs. 5-5, 5-6, 5-13, and 5-14, cause the DOS commands to be displayed on the CRT.

The C, I, O parameters used with MON command are defined as follows:

C—commands to the disk such as OPEN, READ, WRITE, CATALOG, LOCK, UNLOCK, DELETE, SAVE, etc.
I —input from the disk when reading a file from disk to memory.
O—output from memory to disk—writing from memory to disk.

The monitor (MON) command may be used in the deferred execution mode PRINT D$; "MON C,I,O", or the immediate execution mode MON C,I,O in the following formats:

```
 10 REM :CREATE CHR$ VALUES
 20 DIM CHR$(27)
 30 FOR J=1 TO 26: POKE 2054+J,J+128: NEXT J
 40 POKE 2081,30
100 REM :PROGRAM DEMONSTRATING      COMMANDS WITH MONITOR

110 D$=CHR$(4,4)
115 PRINT D$;"MON C"
120 GOSUB 210
130 PRINT "CATALOG"
140 GOSUB 210
150 PRINT "SAVE DUMMY"
160 GOSUB 210
170 PRINT "LOCK DUMMY"
180 GOSUB 210
190 PRINT "CATALOG"
195 PRINT D$;"NOMON C"
200 END
210 PRINT D$;
220 RETURN
```

Fig. 5-13. Integer BASIC program with PRINT D$;, and DOS commands including MON C and NOMON C.

1. MON C—commands to the disk.
2. MON I—input from the disk.
3. MON O—output to the disk.
4. MON C,I—commands to, and input from, the disk.
5. MON C,O—commands to, and output to, the disk.
6. MON I,O—input from, and output to, the disk.
7. MON C,I,O—commands to, input from, and output to, the disk.

The NOMON C, I, O (NO MONITOR) is the reverse of MON C, I, O and turns off all monitor commands. For the MONITOR listings 1 through 7, the same format using NOMON turns the monitor off.

NOMON WITHOUT C, I, O DOES NOT AFFECT THE MONITOR

When programs are saved to disk, they are placed on the disk directory in sequential order, and the user can ensure that the programs are in a specific order. When programs are deleted from the disk, an unused block of space is left on the directory. This space is invisible to the eye, and a CATALOG of the disk does not reveal this unused space. The way to find this space is to do a SAVE to the disk. No program is stored in memory, but a SAVE (program name) is executed. In this example, the name DUMMY will be used to detect the unused space on the disk.

```
 RUN

CATALOG

DISK VOLUME 254

 A 002 DISKINIT
 A 002 FILE STARTER
 A 002 FILE READER
 I 002 FILE STARTER-IB
 I 002 FILE READER-IB
 A 002 CH 5 COMMANDS W/O D$
 A 002 CH 5 COMMANDS WITH D$
 A 002 CH 5 DOS COMMANDS - MON C
 I 003 CH 5 COMMANDS W/O D$-IB
 I 003 CH 5 COMMANDS WITH D$-IB
 I 003 CH 5 DOS COMMANDS - MON C-IB
 A 004 CH 8 CREATE SEQUENTIAL
 A 011 CH 8 UPDATE SEQUENTIAL
 A 008 CH 8 READ SEQUENTIAL
 T 002 TEST FILE

SAVE DUMMY
LOCK DUMMY
CATALOG

DISK VOLUME 254

 A 002 DISKINIT
 A 002 FILE STARTER
 A 002 FILE READER
 I 002 FILE STARTER-IB
 I 002 FILE READER-IB
 A 002 CH 5 COMMANDS W/O D$
 A 002 CH 5 COMMANDS WITH D$
 A 002 CH 5 DOS COMMANDS - MON C
 I 003 CH 5 COMMANDS W/O D$-IB
 I 003 CH 5 COMMANDS WITH D$-IB
 I 003 CH 5 DOS COMMANDS - MON C-IB
 A 004 CH 8 CREATE SEQUENTIAL
 A 011 CH 8 UPDATE SEQUENTIAL
 A 008 CH 8 READ SEQUENTIAL
 T 002 TEST FILE
*I 003 DUMMY

NOMON C
```

Fig. 5-14. RUN of Integer BASIC program with PRINT D$;, and DOS commands including MON C and NOMON C.

When there is an unfilled area on the disk, the next program name saved fills the unused space. The disk catalog looks as follows:

```
A 002 DUPLICATE
A 002 PRIME
A 002 PRIME-1
```

SAVE DUMMY is typed RETURN . DUMMY is placed in the first unused area on the disk.

```
A 002 DUPLICATE
A 002 PRIME
A 002 PRIME-1
A 002 DUMMY
```

If there are no unfilled areas on the disk, DUMMY will be stored as the last program on the catalog.

If there is an unfilled area, before the last program on the disk, the catalog looks as follows:

```
A 002 DUPLICATE
A 002 DUMMY
A 002 PRIME
A 002 PRIME-1
```

To fill the unused space on the disk, follow this procedure.

1. Type DELETE DUMMY RETURN
2. LOAD PRIME RETURN
3. DELETE PRIME RETURN
4. SAVE PRIME RETURN

PRIME now fills the space previously occupied by the name DUMMY. Now there is space between PRIME and PRIME–1, so the routine is followed again to move PRIME–1 into the unfilled space in the disk directory.

SAVING SPACE ON DISKS

Space on disks can be wasted in another way by DOS. A large program, named BIG, with much documentation, and many remarks, filled 60 sectors on a disk. If the large program, BIG, is loaded, and the remarks and the documentation are deleted, and the new program SAVEd to disk, the space requirements are greatly reduced, possibly, to 20 sectors. The documentation and remarks were removed to make the program RUN faster. If the new version of the program, BIG, is saved to disk, DOS will leave all 60 sectors

allocated to the program. If the program now fills only 20 sectors, then 40 sectors are wasted. With the new version of BIG still in memory, DELETE BIG, and then SAVE BIG. The 40 sectors of disk space are now available to store another program.

Disk space is inexpensive, but many unfilled spaces can cause extra disks to be used. With DOS 3.3, the FID program can copy from one disk to another, and squeeze out unused space on disks.

The technique discussed for removing unused space on disks is very good, with one exception. Should power be lost to the computer before the SAVE is complete, the program will be lost. Losing power, partially or completely, is a possibility that must always be considered when saving a program in memory to disk.

chapter 6

Flexibility in DOS Commands

Flexibility is defined as easily bent, not stiff, bending without breaking, or easily adapted to fit various uses, purposes, etc.

Computer programs can be written in a rigid manner, that makes change and adaptation difficult, or they can be written in a flexible way so they can be adapted to similar uses.

The DOS commands can also be used in a rigid or flexible manner. DOS commands are formed by PRINT statements.

For the example used in the rigid program, DUMMY will be the sequential file name. The disk interface card will be in I/O slot No. 6, and disk volume No. 10 is in disk drive No. 2.

RIGID COMMAND

The rigid DOS command to open a sequential file is written as follows:

```
10 PRINT D$; "OPEN DUMMY, S6, D2, V10"
```

The rigid statement can be used in programs successfully, but it is cumbersome. If the program is (1) used for handling sequential files with different names, (2) used on a system that might be reconfigured, or (3) to be

offered for sale to people with systems configured in a different way, a more flexible approach should be used.

FLEXIBLE COMMAND

A flexible series of program statements to handle a sequential file name, slot number, and drive number is with replacement statements.

```
10 F$ = "DUMMY"
20 SL% = 6
30 DR = 2
40 REM : VALUES ARE SET
50 PRINT D$; "OPEN"; F$; ",S"; SL%;" ,D"; DR
```

A more flexible series of program statements to input the sequential file name, slot number, and disk drive is as follows:

```
10 INPUT "FILE NAME IS ";F$
20 INPUT "SLOT NUMBER IS ";S
30 INPUT "DRIVE NUMBER IS ";D
40 PRINT D$; "OPEN"; F$; ",S"; S;" ,D"; D
```

These program statements give flexibility to the program, and allow the user to change the sequential file name, slot number, and disk drive number without rewriting the program. As the user's needs change, and more peripherals are added to the system, the slot and drive numbers change.

COMBINATION RIGID AND FLEXIBLE PROGRAM

Lastly, an even more flexible way to write the program is to use an interactive question to determine if fixed values are to be used to select an often used combination of file name, slot number, and drive number (default values), or if the sequential file name, slot number, and disk drive number are to be input for a seldom used combination. The program using the interactive question to select the combination is the most flexible of the three.

```
100 INPUT "DO YOU WISH TO USE THE DEFAULT VALUES (Y/N)";Q$
110 IF Q$ <> "N" GOTO 160
120 INPUT "FILE NAME IS ";F$
130 INPUT "SLOT NUMBER IS ";S
140 INPUT "DRIVE NUMBER IS ";D
150 GOTO 170
```

```
160 F$ = "DUMMY" : S = 6 : D = 1
170 PRINT D$; "OPEN"; F$;" ,S"; S;" ,D"; D
```

This program allows the user flexibility in that a fixed set of values can easily be accessed, or the values can be input. The replacement statements in line 160 can be changed to accommodate changes in file name, slot number, and drive number. This program presents the best approach to accommodate a rigid method of accessing an often used set of values, and a flexible method to input an infrequently used combination of values.

chapter

Introduction to Files

In this chapter a small company is used as an example for the introduction to files. This company buys manufactured products from a retailer, and sells those products to the public. The business is owned and operated by four people who have no employees. The four owners handle all operations of the company from president to janitor. The business is loosely departmentalized into management and sales, accounting, and shipping and receiving. The business records are contained in a single file cabinet. The business owners are talking about buying a computer system to store and handle the business records, and they want to know how the files are maintained on disks. The illustrations and tables in this chapter are directly related to this business situation.

DATA BASE

A data base, or business records, Fig. 7-1, is a collection of stored information used by the computer system of a particular establishment. A data base is similar to the contents of a large file cabinet used in a business. Within that file cabinet are all files that pertain to business functions. If any portion of the data was destroyed, the business would lose part of the information vital to company operations.

Within the business records are the individual files necessary for daily business transactions.

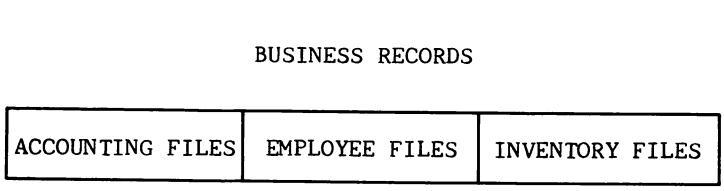


Fig. 7-1. Business records.

FILE

A file, Table 7-1, is an organized collection of records. The relationship between records in a file may be that of a common purpose, format, or data source. In general office procedures, a file cabinet holds documents, usually separated in folders, and indexed according to the particular key. Some files are indexed by a coded number from the least to the greatest, such as social security numbers. Some files will be indexed for mailing by using the ZIP code. Some files will be indexed in alphabetical order.

Table 7-1. Employee File With Four Records

Name	Social Security Number	Department Code	Rate of Pay
Sue Adams	000–00–0000	B4	00.00
Sam Brown	000–00–0000	A2	00.00
John Ho	000–00–0000	C6	00.00
Helen Zebronski	000–00–0000	D8	00.00

Some files will be updated daily, such as daily sales, and accounts receivable. Some files will be updated weekly, such as payroll.

In this book, two methods of access to files will be discussed: sequential access, and random, or direct, access.

SEQUENTIAL ACCESS FILES

Sequential access files, Fig. 7-2, are files consisting of records of variable length in which data can be accessed in the sequence in which it is stored in the file. An item, or record, can only be examined, or processed, after all preceding records have been accessed.

As an example of a sequential file, suppose you were going to look up the word *radio* in the dictionary. To look up radio in a sequential manner, every word from the first word (letter A in this dictionary) had to be scanned before radio was discovered. This would be a slow, tedious task just to look up one word.

COMPOSITION OF A SEQUENTIALLY ACCESSED RECORD

CHARACTER:	J	O	H	N	sp	H	O	cr	0	0	0	-	0	0	-	0	0	0	0	cr	C	6	cr	0	0	.	0	0	cr
ASCII:	74	79	72	78	32	72	79	13	48	48	48	45	48	48	45	48	48	48	48	13	67	54	13	48	48	46	48	48	13
FIELD SIZE:				8										12								3				6			

CHARACTER:	H	E	L	E	N	sp	Z	E	B	R	O	N	S	K	I	cr	0	0	0	-	0	0	-	0	0	0	0	cr	D	8	cr	0	0	.	0	0	cr
ASCII:	72	69	76	69	77	32	90	69	66	82	79	78	83	75	73	13	48	48	48	45	48	48	45	48	48	48	48	13	68	56	13	48	48	46	48	48	13
FIELD SIZE:									16														12							3				6			

ASCII 13 is a carriage return
Fields of sequenctial file records are of different lengths which saves storage space but makes access time to different records in the file relatively slow.

Fig. 7-2. Composition of a sequentially accessed record.

A sequential access file has slow access time from one record to any other record, but takes a minimum of storage space for each record.

Within the file cabinet are records of employee information that will contain the employee's name, social security number, departmental code, and pay scale (Table 7-2). There are parts of these records that change frequently, but the majority of the record remains intact. These records must be easily updated without destroying the entire record contents. This type of file may be randomly, or directly, accessed.

Table 7-2. Employee Record With Four Fields

Name	Social Security Number	Department Code	Rate of Pay
John Ho	000–00–0000	C6	00.00

RANDOM ACCESS FILES

A random access file, or direct access file, Fig. 7-3, is of specific record length and is designed to give a relatively constant (fast) access time regardless of the record previously accessed. An employee file of 10,000 records randomly accessed should have a constant access time if the first record was accessed, and then the next record accessed was the last record in the file. Randomly accessed records have a constant record length, and a constant access time, but take more memory space than sequential access records.

RECORDS AND UPDATING RECORDS

There are other records, such as the MASTER INVENTORY FILE, that should be updated daily. In most cases this file is not updated directly. In case hardware problems or power failure occurred, a direct updating of the MASTER INVENTORY FILE could cause destruction of the file. A NEW INVENTORY FILE is then created, so that the CURRENT INVENTORY FILE will not be lost, should an accident bring down the system during the update.

A temporary file is a file used to store information for file protection, or other reasons, such as generating reports that will be deleted when the function is completed.

Files are classified according to the way they are built or used, and the type may be either sequential access, or random access. The four file classifications are: (1) source, (2) destination, (3) source-destination, or (4) piggyback.

COMPOSITION OF A RANDOMLY ACCESSED RECORD 40 BYTES LONG

CHARACTER:	J	O	H	N	sp	H	O	cr	0	0	0	-	0	0	0	0	cr	C	6	cr	0	0	.	0	0	0	0	0	0	0	0	0
ASCII:	74	79	72	78	32	72	79	13	48	48	48	45	48	48	48	48	13	67	54	13	48	48	46	48	48	48	48	48	48	48	48	48
FIELD SIZE:					8								12						3				6					11				

CHARACTER:	H	E	L	E	N	sp	Z	E	B	R	O	N	S	K	I	cr	0	0	0	-	0	0	-	0	0	0	0	cr	D	8	cr	0	0	.	0	0	cr	0	0	0
ASCII:	72	69	76	69	77	32	90	69	66	82	79	78	83	75	73	13	48	48	48	45	48	48	45	48	48	48	48	13	68	56	13	48	48	46	48	48	13	48	48	48
FIELD SIZE:							16													12									3					6				3		

ASCII 13 is a carriage return
Randomly accessed records take more storage space, but the access time from one record to any other record in the file is very fast.

Fig. 7-3. Composition of a randomly accessed record 40 bytes long.

A source file is a file that is used merely for input. In this example, a CURRENT MASTER FILE has yesterday's information stored on it. A NEW MASTER FILE will have today's information stored on it. To protect the company records, the CURRENT MASTER FILE is read. Today's transactions are read to update the CURRENT MASTER FILE and to create a NEW MASTER FILE. The NEW MASTER FILE then becomes a source file when tomorrow's update is to be done. Each day a NEW MASTER FILE is created, and the next day the NEW MASTER becomes the CURRENT MASTER. The CURRENT MASTER FILE is protected by creating a NEW MASTER FILE in case of hardware problems, if the processor loses power, or if a disk or tape should be defective.

A destination file is used only for output. When a destination file is built, it becomes a NEW MASTER FILE. The NEW MASTER FILE is only used for output. The CURRENT MASTER FILE and the NEW MASTER FILE are both retained for a specific period of time to protect the records of the business. In many businesses, it is standard practice to keep a copy of both the CURRENT MASTER FILE, and the NEW MASTER FILE for a period of five days before they are discarded.

A source-destination file is a file used for input and output. The records on this file can be read, changed, and written back to the file.

A random file is an example of a source-destination file. It takes less computer hardware to create and maintain this type of file because a random access file is a single file operation, whereas a sequential file is a double file operation, one sequential file is read from, and the second sequential file is written to. If the processer loses power, or the disk or tape is destroyed, when updating a random access source destination file, it has to be rebuilt from the beginning. There must be a copy made of the CURRENT MASTER FILE as added protection.

A piggyback file is a file built one section at a time. When a computer has only 48K memory, and necessary storage to build the file requires 80K memory, the file must be built in sections compatible with available memory. A program is written to create a catalog of the program names stored on, for example, 100 disks. Each disk has to be read into memory separately. The program names are properly sequenced on a master catalog file until all the program names on the 100 disks have been cataloged. The piggyback file is created by reading and cataloging each disk separately.

Files are composed of records. A record, Fig. 7-2, is a collection of data items. An employee file would include employee records. An employee record would include employee name, social security number, department code, etc.

NAME FIELD OF EMPLOYEE RECORD

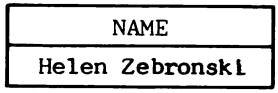


Fig. 7-4. Name field of employee record.

The record count is the number of records in each file. The record count uses a counter (C = C + 1) within the program to number the records as the file is built. The record count is also a method to retrieve a record in a randomly accessed file.

Table 7-3. ASCII Character Codes

Code		Char	Code		Char	Code		Char	Code		Char
Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
0	00	NUL	32	20	SP	64	40	@	96	60	
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	—	127	7F	DEL

Table 7-4. ASCII Character Codes—Binary

Bit Numbers →								0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_7 ↓	b_6 ↓	b_5 ↓	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
			0	0	0	0	0	NUL	DLE	SP	0	@	p		p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	‖	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
			0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
			1	0	1	1	B	VT	ESC	+	;	K	[	k	{
			1	1	0	0	C	FF	FS	,	<	L	\	l	|
			1	1	0	1	D	CR	GS	−	=	M	]	m	}
			1	1	1	0	E	SO	RS	.	>	N	^	n	~
			1	1	1	1	F	SI	US	/	?	O	−	o	DEL

Each record is divided into fields. A field, Fig. 7-4 and Table 7-4, is a subdivision of a record.

Files are encoded on disks using the ASCII code, Tables 1-4, 1-5, 7-3, 7-4 and Figs. 7-2 and 7-3. ASCII is the acronym for AMERICAN STANDARD CODE for INFORMATION INTERCHANGE. Tables 7-3 and 7-4 are alpha, numeric, and special characters and are represented by decimal, hexadecimal, and binary numbers.

chapter 8

Sequential Files

OVERVIEW OF SEQUENTIAL STORAGE

Sequential storage is a technique in which stored records are available on a one after another basis.

The sequential file is the oldest and most primitive of all file structures. Its design is strongly related to the predisk period when only punched cards, punched tape, and magnetic tape were available for bulk storage. A sequential file is stored in a set of storage blocks, or access blocks. Generally, the data is placed in records of unequal length so no storage space is wasted, but the unequal length makes for slow access time to different parts of the file.

The sequential file has no directory to locate individual records. The records are generally organized in some logical order on a specific field. The records may be placed from A to Z, according to name, or from low order to high order social security numbers, or from low order to high order zip code numbers. Before the records are placed in the file they may be sorted according to the desired key.

At times a file is considered to be sequentially organized even though it is not ordered to any key. If the date an item is received is the important factor, then the date of receipt is the key to the file. The entries can then be added at the end of the file without moving any records, and the file will be in the proper sequence.

The main advantages of a sequential file are: (1) it is easy to create, and (2) it provides fast access when going from one record to the next record.

The main disadvantages of a sequential file are: (1) it is difficult to update, and (2) access to nonadjacent records is extremely slow.

A microcomputer system that has only a cassette tape recorder for bulk storage has no choice except to use a sequential file.

Sequential files are frequently used in financial institutions where customer records may contain one or more account items. If each account had to be contained in a fixed amount of storage space, it would be difficult to judge the correct amount of space to reserve for the accounts. With the sequential file the account history items can be read until the end of the account record is reached. In this way, no specific amount of storage space has to be reserved.

Manuscripts, or books, are types of sequential files, in that one sentence follows another. When corrections, insertions, and deletions are made, the file must be changed and updated.

Retrieval of a specific record from a sequential access file is by reading all the records until a match is found, or until the end of the file is reached, and no match is found. To retrieve a single record, the entire file may have to be scanned. If another record is to be retrieved, the file is reread, and the search begins again. The programs and runs, Figs. 8-1 through 8-10, handle sequential files.

Fig. 8-2 is a program to create a special record to mark the end of a sequential text file. Fig. 8-3 is a run of the program in Fig. 8-2, and shows the end of file record that is created. The only purpose of this program is to create the end of file record.

```
END OF THE FILE RECORD

   NAME = END RECORD
   SOCIAL SECURITY NUMBER = 000-00-0000
   DEPARTMENT CODE = 00
   PAY RATE = 00.00
```

When the end of the file has been established, then the file can be built, and the end record can be placed as necessary to increase the size of the file. The program in Fig. 8-2 is written to set up the four fields of the employee record as discussed in Chapter 7. The end of the file record must be the last record in the file so the program will know when and where to stop reading the file. When the last record, or end of file, is reached, the DOS command PRINT D$; "CLOSE" causes the file to be closed. The CLOSE is a mandatory command that must be activated if the file is prop-

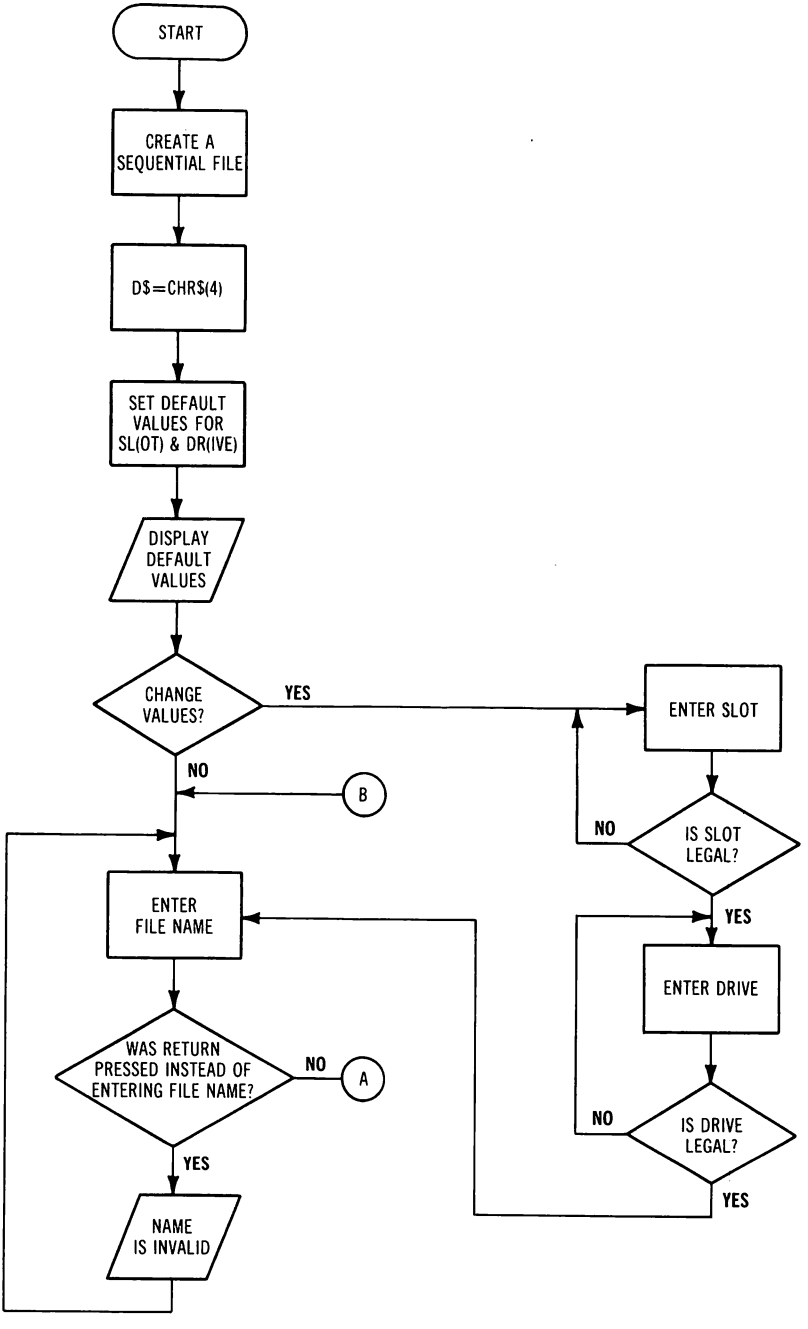


Fig. 8-1. Flowchart of program to create a sequential text file.

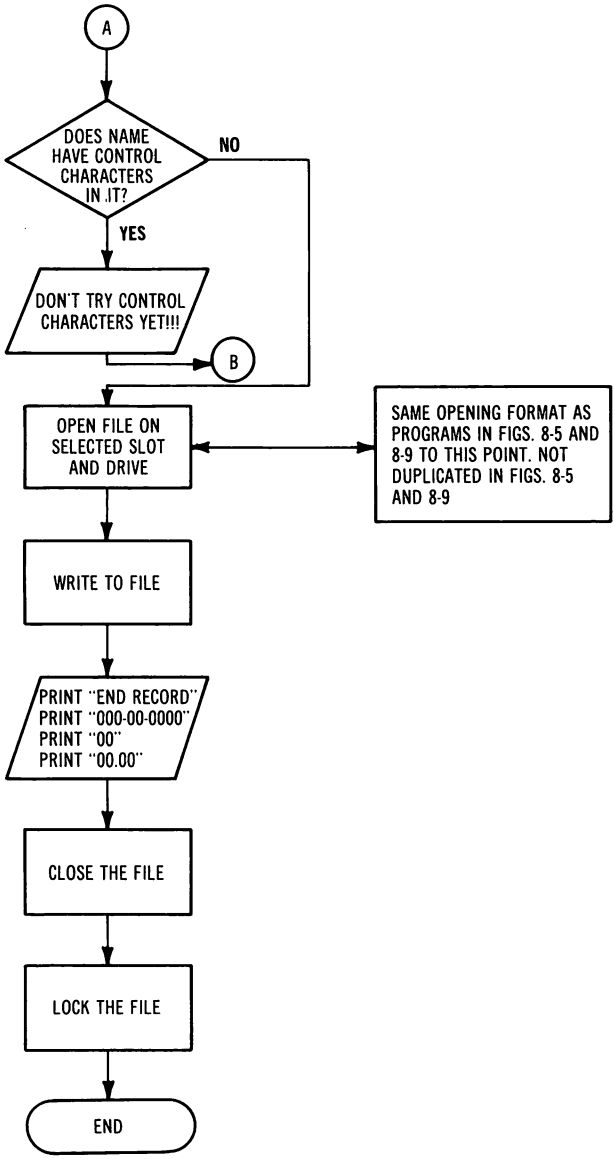


Fig. 8-1.—*Cont.* **Flowchart of program to create a sequential text file.**

erly closed. The command to LOCK TEST-RECORDS is an optional command, but is issued to prevent the user from accidentally writing to the file and destroying its contents. When the file is locked, it can only be read from—it cannot be written to.

```
100  REM :PROGRAM TO CREATE A    SEQUENTIAL FILE
110 D$ =  CHR$ (4)
120 SL = 6: REM :SLOT IS #6
130 DR = 1: REM :DRIVE IS #1
140  HOME : VTAB 3
150  PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
160  PRINT
170  INPUT "CHANGE THESE VALUES(Y OR N)? ";QU$
180  PRINT
190  IF QU$ <  > "Y" THEN 240
200  INPUT "ENTER SLOT (1-7) ?";SL
210  IF SL < 1 OR SL > 7 THEN 200
220  INPUT "ENTER DRIVE (1-2) ?";DR
230  IF DR <  > 1 AND DR <  > 2 THEN 220
240  PRINT : INPUT " ENTER FILE NAME ?";F$
250  IF  LEN (F$) = 0 THEN  PRINT : PRINT "*** NAME IS
      INVALID ***": GOTO 240
260  FOR J = 1 TO  LEN (F$): IF  ASC ( MID$ (F$,J,1)) <
     32 THEN  PRINT : PRINT "*** PLEASE DON'T TRY CONT
     ROL CHARA-": PRINT "CTERS YET !": GOTO 240
270  NEXT J: PRINT : PRINT "!!! THE NAME IS GOOD"
280  PRINT D$;"OPEN ";F$;",S";SL;",D";DR
290  PRINT D$;"WRITE ";F$
300  PRINT "END RECORD": PRINT "000-00-0000": PRINT "0
     0": PRINT "00.00"
310  PRINT D$;"CLOSE"
320  PRINT D$;"LOCK ";F$;",S";SL;",D";DR
330  END
```

Fig. 8-2. Program to create a sequential text file in Applesoft language.

```
 MON C, I, O

]RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N                    THIS NORMALLY SHOWS ON
                                                  THE SCREEN

 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
OPEN TEST-RECORDS,S6,D1
WRITE TEST-RECORDS
END RECORD                                        THIS DOES NOT SHOW ON
000-00-0000                                       THE SCREEN WITHOUT
00                                                MON C,I,O
00.00
CLOSE
LOCK TEST-RECORDS,S6,D1

]PR#0
```

Fig. 8-3. RUN of the program in Fig. 8-2, create a sequential text file.

The employee file was created to be used in practical business applications. Such applications would include reading the employee file into a payroll program to create the payroll and print out the pay checks.

Fig. 8-5 is a program to update a sequential text file in that it can:

(1) change a record, (2) delete a record, (3) insert a record, or (4) add records at the end of the file (before the end of file record). Fig. 8-9 is a program that reads (lists) the records to the CRT. In Figs. 8-3, 8-7, and 8-10 the name used for the created file is TEST-RECORDS.

CREATE A SEQUENTIAL TEXT FILE

Fig. 8-1 is a flowchart of the program to create a sequential text file, and Fig. 8-2 is an Applesoft program to create a sequential text file. Control D is initialized in line 110 as D$ = CHR$ (4). Lines 120 through 280 of this flexible program set up the default values input/output slot number as #6, and the disk drive as #1. If you are going to use the default values in this program, make sure they are the proper values to match your computer configuration.

Lines 120 through 280 are identical lines of initialization that appear in Fig. 8-2, a program to create a sequential text file; Fig. 8-5, a program to update a sequential text file; and Fig. 8-9, a program to read (list) a sequential text file.

The interactive question in line 190 allows use of the default values, or they can be changed.

The check in line 210 prevents any slot number less than 1 from being accepted, or any slot number greater than 7 from being accepted. Slot zero is restricted to the use of the language card. The Apple has 7 slots usable for input/output devices.

Line 230 is a check so that no number less than 1 is accepted for a disk drive number, and no number greater than 2 is accepted as a disk drive number. The disk interface card can handle only two disk drives. If your system has only one disk drive in the selected slot, entering 2 could cause the program to fail, and lock up your system.

Line 240 requires that the file name be input. The file name input is TEST-RECORDS, and is used as the created file in the programs in Figs. 8-3, 8-7, and 8-10.

Line 250 checks to see if a file name is input. If no file name is input, and RETURN is pressed, the error message *** NAME IS INVALID ***, is printed on the CRT and control is returned to line 240, to allow the user to input a valid file name.

Line 260 is a check that shows the user if a control character has been placed in the file name, and returns control to line 240 to input a file name without a control character.

To use the file, the file name must be typed exactly as it was when

created. Invisible control characters typed in a file name will give you a big headache if you forget what character was typed, or its position in the file name. This is similar to the situation when you have a combination lock, but forget its combination. The lock is useless.

If you wish to place control characters in the file name for protection of your files, delete lines 260 and 270 from the program.

If the file name passes the checks in lines 250 and 260, line 270 prints out that the file name is valid.

Line 280 opens the file TEST-RECORDS, and creates the end record in the file.

Lines 290 through 320 tell the DOS to send all output from PRINT statements, to the file TEST-RECORDS, and prints the end of file record to the disk. This end of file record indicates to the program where the file ends. (PRINT D$; "WRITE", and PRINT F$ are the combination to place a value on the file. PRINT D$, and PRINT are two entirely different commands.)

```
310 PRINT D$;"CLOSE"
```

When CLOSE is used with no file name, it closes all files that are open. Data is not lost from a file that is being read if that file is not closed; however, the preferred procedure is to close all files.

```
320 PRINT D$;"LOCK";F$;",S";SL;",D";DR
```

This statement locks the file so the records cannot be destroyed, either by writing over them, or deleting the file. Line

```
330 END
```

ends the program.

UPDATE A SEQUENTIAL TEXT FILE

Fig. 8-4 is a flowchart of the program to update a sequential text file, and Fig. 8-5 is the program to update a sequential text file. It is the second program in a series of three programs dealing with sequential text files. The program in Fig. 8-5 utilizes the sequential file, TEST-RECORDS, that was created by the program in Fig. 8-2. Lines 110 through 280 are the same as in Fig. 8-2. Line

```
290 PRINT D$;"OPEN DUMMY ,S";SL;",D";DR
```

opens a dummy file on the disk. In the update program, the file TEST-RECORDS will be read, and the records will be written to a DUMMY file.

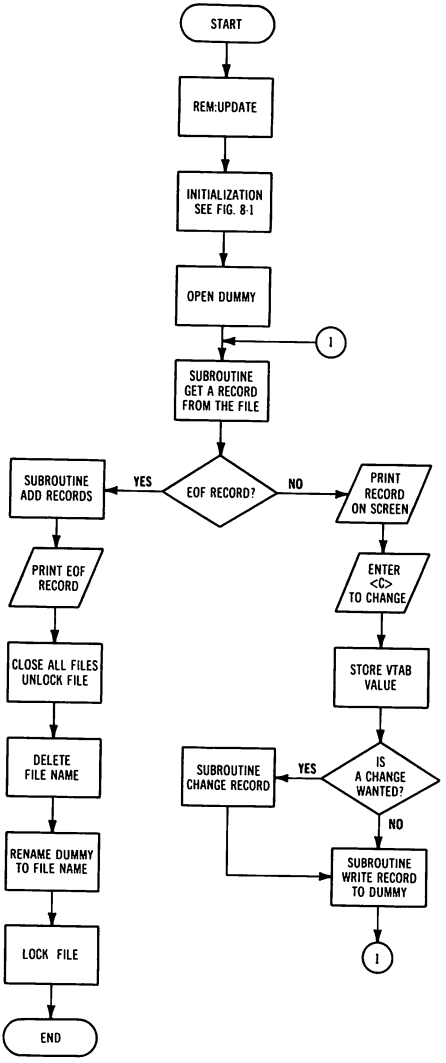
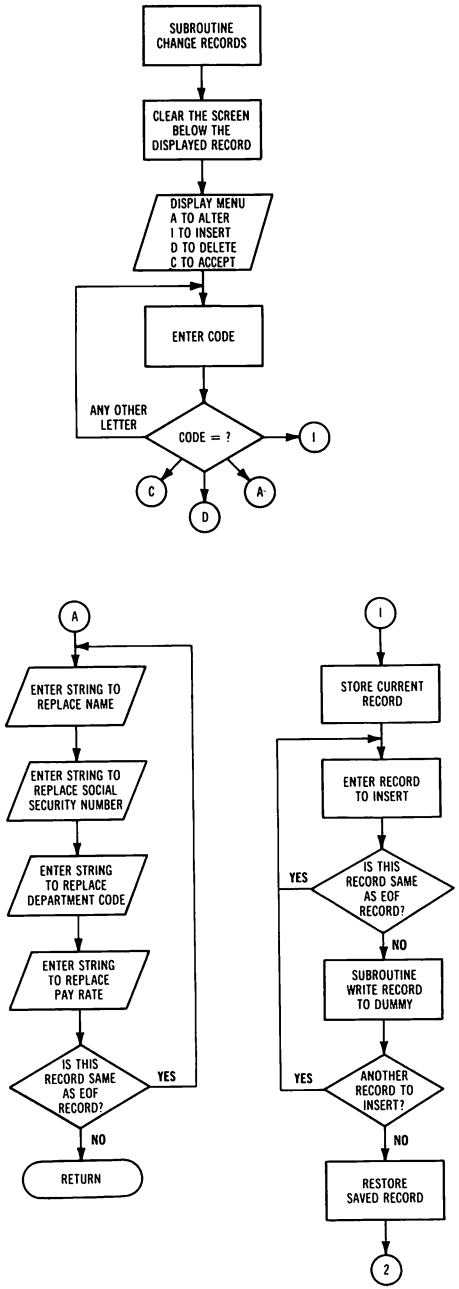


Fig. 8-4. Flowchart of program



(cont. next page)

to update a sequential text file.

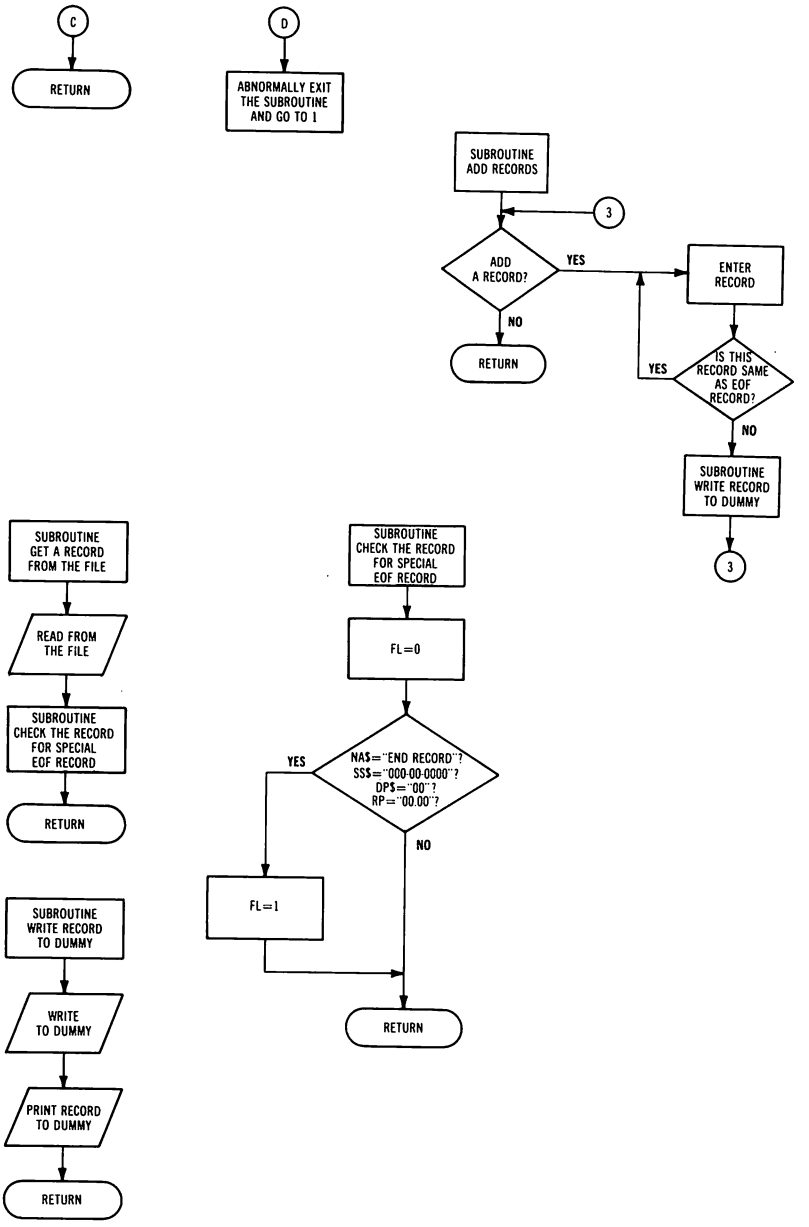


Fig. 8-4—*cont.* Flowchart of program to update a sequential text file.

```
100  REM :PROGRAM TO UPDATE A    SEQUENTIAL FILE
110 D$ =  CHR$ (4)
120 SL = 6: REM :SLOT # IS 6
130 DR = 1: REM :DRIVE # IS 1
140  HOME : VTAB 3
150  PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
160  PRINT
170  INPUT "CHANGE THESE VALUES(Y OR N)? ";QU$
180  PRINT
190  IF QU$ <  > "Y" THEN 240
200  INPUT "ENTER SLOT (1-7) ?";SL
210  IF SL < 1 OR SL > 7 THEN 200
220  INPUT "ENTER DRIVE (1-2) ?";DR
230  IF DR <  > 1 AND DR <  > 2 THEN 220
240  PRINT : INPUT " ENTER FILE NAME ?";F$
250  IF  LEN (F$) = 0 THEN  PRINT : PRINT "*** NAME IS
      INVALID ***": GOTO 240
260  FOR J = 1 TO  LEN (F$): IF  ASC ( MID$ (F$,J,1)) <
     32 THEN  PRINT : PRINT "*** PLEASE DON'T TRY CONT
     ROL CHARA-": PRINT "CTERS YET !": GOTO 240
270  NEXT J: PRINT : PRINT "!!! THE NAME IS GOOD"
280  PRINT D$;"OPEN ";F$;",S";SL;",D";DR
290  PRINT D$;"OPEN DUMMY,S";SL;",D";DR
300  GOSUB 520
310  IF FL THEN 410
320  HOME
330  PRINT "NAME =";  SPC( 4);NA$
340  PRINT : PRINT "S SEC NUM = ";SS$
350  PRINT : PRINT "DEPT   =  ";DP$
360  PRINT : PRINT "PAY RATE= ";RP
370  PRINT : INPUT "TO CHANGE ENTER <C>";QU$
380 CP =  PEEK (37) - 1: IF QU$ = "C" THEN  GOSUB 730
390  GOSUB 560
400  GOTO 300
410  GOSUB 600
420 NA$ = "END RECORD":SS$ = "000-00-0000":DP$ = "00":
     RP = 0: GOSUB 560
430  PRINT D$;"CLOSE"
440  PRINT D$;"UNLOCK ";F$;",S";SL;",D";DR
450  PRINT D$;"DELETE ";F$;",S";SL;",D";DR
460  PRINT D$;"RENAME DUMMY,";F$
470  PRINT D$;"LOCK ";F$;",S";SL;",D";DR
480  END
490 FL = 0
500  IF NA$ = "END RECORD" AND SS$ = "000-00-0000" AND
     DP$ = "00" AND RP = 0 THEN FL = 1
510  RETURN
520  PRINT D$;"READ ";F$
530  INPUT NA$,SS$,DP$,RP
540  PRINT D$;"IN#0": GOSUB 490
550  RETURN
560  PRINT D$;"WRITE DUMMY"
570  PRINT NA$: PRINT SS$: PRINT DP$: PRINT RP
580  PRINT D$;"PR#0"
590  RETURN
600  INPUT "DO YOU WANT TO ADD MORE RECORDS ?";QU$
610  IF QU$ <  > "Y" THEN  RETURN
```

Fig. 8-5. Program to update a sequential text file in Applesoft.

```
620  HOME
630  GOSUB 690:NA$ = QU$
640  GOSUB 700:SS$ = QU$
650  GOSUB 710:DP$ = QU$
660  GOSUB 720:RP =  VAL (QU$)
670  GOSUB 490: IF FL THEN  PRINT "THIS IS AN ILLEGAL
     RECORD !!!": FOR J = 1 TO 500: NEXT J: PRINT  CHR$
     (7): GOTO 620
680  GOSUB 560: GOTO 600
690  INPUT "NAME = ";QU$: RETURN
700  INPUT "S SEC NUM = ";QU$: RETURN
710  INPUT "DEPT.  = ";QU$: RETURN
720  INPUT "RATE  = ";QU$: RETURN
730  POKE 37,CP: CALL  - 958: PRINT "ENTER :": PRINT :
      HTAB 7: PRINT "A TO ALTER"
740  HTAB 7: PRINT "I TO INSERT NEW RECORD"
750  HTAB 7: PRINT "D TO DELETE ITEM"
760  HTAB 7: PRINT "C TO ACCEPT ITEM AS IS"
770  PRINT : PRINT : INPUT "ENTER CODE ?";QU$
780  ON ((QU$ = "I") + (QU$ = "A") * 2 + (QU$ = "D") *
     3 + (QU$ = "C") * 4) GOTO 790,980,1070,1060: GOTO
     730
790 L$ = NA$ + SS$ + DP$ +  STR$ (RP)
800 L1 =  LEN (NA$):L2 =  LEN (SS$):L3 =  LEN (DP$):
810  POKE 37,CP: CALL  - 958
820  GOSUB 690:NA$ = QU$
830  GOSUB 700:SS$ = QU$
840  GOSUB 710:DP$ = QU$
850  GOSUB 720:RP =  VAL (QU$)
860  GOSUB 490: IF FL THEN  PRINT "THIS IS AN ILLEGAL
     RECORD !!!": FOR J = 1 TO 500: NEXT J: PRINT  CHR$
     (7): GOTO 810
870  GOSUB 560: INPUT "INSERT ANOTHER RECORD (Y OR N)
     ?";QU$
880  IF QU$ = "Y" THEN 810
890 L0 =  LEN (L$)
900 NA$ =  LEFT$ (L$,L1)
910 L$ =  RIGHT$ (L$,L0 - L1):L0 = L0 - L1
920 SS$ =  LEFT$ (L$,L2)
930 L$ =  RIGHT$ (L$,L0 - L2):L0 = L0 - L2
940 DP$ =  LEFT$ (L$,L3)
950 L$ =  RIGHT$ (L$,L0 - L3):L0 = L0 - L3
960 RP =  VAL (L$)
970  GOTO 730
980  POKE 37,CP: CALL  - 958: PRINT "ENTER NEW VALUE O
     R 'RETURN'"
990  PRINT "TO LEAVE CURRENT VALUE UNCHANGED"
1000  PRINT : GOSUB 690: IF  LEN (QU$) > 0 THEN NA$ =
     QU$
1020  GOSUB 700: IF  LEN (QU$) > 0 THEN SS$ = QU$
1030  GOSUB 710: IF  LEN (QU$) > 0 THEN DP$ = QU$
1040  GOSUB 720: IF  LEN (QU$) > 0 THEN RP =  VAL (QU$
1050  GOSUB 490: IF FL THEN  PRINT "ILLEGAL RECORD !!!
     ": FOR J = 1 TO 500: NEXT J: PRINT  CHR$ (7): GOTO
     980
1060  RETURN
1070  POP : GOTO 300
```

Fig. 8-5—*cont.* Program to update a sequential text file in Applesoft.

This includes old records that have been altered, records that have been inserted into the file, or records that have been added to the end of the file, after the end of file record has been read. This is called a file to file transfer. After the program has been run, and all file maintenance has been completed, the program goes to line 420 to print the end of file record on the file. The file is closed at line 430. Line

```
440 PRINT D$; "UNLOCK ";F$;",S";SL;",D";DR
```

unlocks the file name TEST-RECORDS, slot #6, drive #1. Line

```
450 PRINT D$; "DELETE ";F$;",S";SL; ",D";DR
```

deletes the file named TEST-RECORDS in slot #6, drive #1. Line

```
460 PRINT D$;"RENAME DUMMY,";F$
```

```
 LOGIC STATEMENT      MULTIPLIER  TOTAL  ON GOTO DESTINATION L

*QU$="I"=(INSERT)          1         1          790
*QU$="A"=(ALTER)           2         2          980
*QU$="D"=(DELETE)          3         3          1070
*QU$="C"=(ACCEPT AS IS)    4         4          1060


*IF QU$="I" IS TRUE (1), ALL OTHER VALUES ARE FALSE (0)
*IF QU$="A" IS TRUE (1), ALL OTHER VALUES ARE FALSE (0)
*IF QU$="D" IS TRUE (1), ALL OTHER VALUES ARE FALSE (0)
*IF QU$="C" IS TRUE (1), ALL OTHER VALUES ARE FALSE (0)
```

Fig. 8-6. Logic of transfer on QU$ GOTO.

renames the file created under the name of DUMMY, to TEST-RECORDS. The DUMMY file was created during the file maintenance of the TEST-RECORDS file. The file named DUMMY, into which all updated records were placed, is renamed TEST-RECORDS. If much file maintenance was done, the new file TEST-RECORDS is a very different, and more current, file than the original TEST-RECORDS. Line

```
470 PRINT D$;"LOCK ";F$;",S";SL;",D";DR
```

locks TEST-RECORDS in slot #6, drive #1.

On a one disk drive system, one important fact to realize from creating a dummy file is that only half of the disk storage space can be used by the file to file transfer method. The DUMMY file takes half the disk, and the TEST-RECORDS file uses the other half. With a DOS 3.3, 146K bytes storage space this would allow two files, each with approximately 1500 records, with an average of 40 bytes per record.

On a two disk drive system, disk drive #1 could be used to read the TEST-RECORDS file, and disk drive #2 could be used to write the records to the DUMMY file.

After the DUMMY file is opened in line 290, the program defaults to line 300 which is GOSUB 520.

The subroutine that begins at line 520 READS a record from the file TEST-RECORDS stored on disk.

Line 530 INPUT NA$,SS$,DP$,RP is the method used to input the name, social security number, department code, and rate of pay into memory. Line

```
540 PRINT D$;"IN#0":GOSUB 490
```

tells DOS to change the input condition from the file to the keyboard. If a disk has been read from, it is the default unit for input. On the next input instruction, the computer will reference the default unit, unless the IN#0 condition is set. Each time the input/output source is changed, the default unit is changed. When input is done with a message in a quote field, this is actually a combination PRINT and INPUT.

```
PRINT D$;"WRITE";F$
PRINT NA$,SS$,DP$,RP
INPUT "ARE YOU GOING TO DISK";QU$
```

(ARE YOU GOING TO DISK is never seen unless MON O is turned on.)

If the print direction is set to disk, the interactive question will be put on the disk file. To prevent the interactive question from going to disk, set the print direction to PR#0 (the input direction should have been set to IN#0 since the input was from the keyboard). Always tell the computer system exactly where the INPUT and PRINT should go.

In the UPDATE program, the input direction is set to TEST-RECORDS file by the command

```
PRINT D$;"READ ";F$
```

in order to read the record into memory, lines 520, and 530. The INPUT device is then set to IN#0 to ensure that the keyboard will be accessed for any INPUT command. This is because there are so many interactive questions in the program.

In the UPDATE program, the print direction is set to the DUMMY file by the command

```
PRINT D$;"WRITE DUMMY"
```

to ensure that the record is written to the DUMMY file, lines 560 through

590. The output device is set to PR#0 to ensure that the default device is the screen. This is because of the interactive questions that are so prevalent in the program.

From the subroutine beginning at line 520, a subroutine is called from line 540—GOSUB 490.

The subroutine beginning at line 490 checks to see if the record just read in from the file, or the record just input by the user, is the same as the end of file record. The subroutine at line 490 sets the FL(ag) = 0, and compares the record with the special end of file record. If FL = 0, the end of file has not been read, and the subroutine returns to line 550, RETURN, which causes the program to jump to line 310. If the record just read in is the end of file record, the FL(ag) value is set to 1, and line 510 RETURN is executed. The RETURN causes the program to jump to line 550, which RETURNS to line 310—IF FL THEN 410.

```
310 IF FL THEN 410
```

is an application of logical values in Applesoft. In the subroutine beginning at line 490, the end of file record has either been read (TRUE—FL = 1), or it hasn't been read (FALSE—FL = 0). The more common structure of line 310 would be, IF FL = 1 THEN 410. The use of logic in programming is a more sophisticated approach.

```
310 IF FL THEN 410
```

If the flag value is 1, and the statement is true, it means the end of the file record has been read. The program branches to line 410 GOSUB 600. The subroutine which begins at line 600 allows the user to add more records at the end of the file.

```
600 INPUT "DO YOU WANT TO ADD MORE RECORDS ?";QU$
610 IF QU$ <> "Y" THEN RETURN
```

uses reverse logic in this decision statement. If the user presses any key but "Y", the check is true, and it returns. If the user presses "Y" RETURN the check is false, and it defaults to the lines 620 through 680 to build a new record.

```
620 HOME
```

clears the screen.

```
630 GOSUB 690 : NA$ = QU$
```

The program jumps to the subroutine at line 690 which places "NAME = " (Fig. 8-7) in the top left corner of the screen. The user inputs the

```
]MON C, I, O

]RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N


 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
OPEN TEST-RECORDS,S6,D1
OPEN DUMMY,S6,D1
READ TEST-RECORDS
?END RECORD
??000-00-0000
??00
??00.00
IN#0
DO YOU WANT TO ADD MORE RECORDS ?Y
NAME = JOHN HO
S SEC NUM = 000-00-0000
DEPT.  = XYZ6
RATE  = 9.60
WRITE DUMMY
JOHN HO
000-00-0000
XYZ6
9.6
PR#0


]MON C, I, O

]RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N


 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
OPEN TEST-RECORDS,S6,D1
OPEN DUMMY,S6,D1
READ TEST-RECORDS
?JOHN HO
```

Fig. 8-7. RUN of the program in Fig. 8-5, program to update a sequential text file.

```
??000-00-0000
??C6
??9.6
IN#0
NAME =     JOHN HO

S SEC NUM = 000-00-0000

DEPT    =  C6

PAY RATE= 9.6

TO CHANGE ENTER <C>C
ENTER :

       A TO ALTER
       I TO INSERT NEW RECORD
       D TO DELETE ITEM
       C TO ACCEPT ITEM AS IS


ENTER CODE ?A
ENTER NEW VALUE OR 'RETURN'
TO LEAVE CURRENT VALUE UNCHANGED

NAME =
S SEC NUM =
DEPT.   = C6
RATE   =
WRITE DUMMY
JOHN HO
000-00-0000
C6
9.6
PR#0
```

Fig. 8-7—*cont.* RUN of the program in Fig. 8-5, program to update a sequential text file.

employee's name in the blank space after "NAME = ". The name is placed in QU$, and when the program returns to the second statement in line 630 the value held in QU$ is assigned to NA$.

```
640 GOSUB 700 : SS$ = QU$
```

The program jumps to the subroutine that begins at line 700, and prints "S SEC NUM = " on the screen. The user inputs the employee's social security number into QU$, and the program returns to the second statement in line 640. The value held in QU$ is then assigned to SS$.

```
650 GOSUB 710 : DP$ = QU$
```

The program jumps to the subroutine that begins at line 710, and prints DEPT. = on the screen. The user inputs the employee's department designation into QU$. When the program returns to the second statement at line 650, the value in QU$ is assigned to DP$.

```
660 GOSUB 720 : RP = VAL(QU$)
```

The program jumps to the subroutine that begins at line 720 and prints "RATE = " on the screen. The user inputs the employee's rate of pay into QU$. When the program returns to the second statement in line 660, the value in the alphanumeric QU$ is assigned to the numeric RP variable—RP = VAL(QU$). VAL is the basic word that transfers an alphanumeric string value into a numeric value.

```
670 GOSUB 490 : IF FL THEN PRINT "THIS IS AN ILLEGAL RECORD!!!" : FOR
  J = 1 TO 500 : NEXT J : PRINT CHR$(7) : GOTO 620
680 GOSUB 560 : GOTO 600
```

GOSUB 490 is a subroutine that compares the record input with the end of the file record. If the record just input and the end of the file record are the same, the flag value is set to 1 (TRUE), and the program returns to the second statement in line 670. If FL = 1, the program will not accept two end records, and prints out "THIS IS AN ILLEGAL RECORD!!!".

The pause loop increments 500 times (FOR J = 1 TO 500 : NEXT J), and the computer bell clangs (PRINT CHR$(7)), to let the world know the user has made a boo boo. The program jumps to line 620 (GOTO 620), to allow the user another chance to input a correct record.

In line 670, if the FL value returned from the subroutine that begins at line 490 is zero, the record just input is acceptable, and the program defaults to line 680.

```
680 GOSUB 560 : GOTO 600
```

The subroutine that begins at line 560 writes the acceptable record into the DUMMY file. The return from the subroutine (560–590) causes the program to jump to line 600, to ask the question, "DO YOU WANT TO ADD MORE RECORDS?"

```
310 IF FL THEN 410
```

If FL = 0, the statement is false, and the program defaults to the series of lines 330 through 360 to print the record just read from the file, on the CRT. Since FL = 0, the record just read cannot be the end of the file record.

Line 370 (PRINT "TO CHANGE ENTER <C>";QU$) asks the question, do you want to change this record, and/or do you want to change the file itself?

In the portion of the program beginning at line 730 are four options: (1) A—to alter a record, which causes a single record to be changed, (2) I—insert a new record, which causes the file to be changed, (3) D—to delete a record, which causes the file to be changed, and (4) C—accept the record as is, which leaves the file unchanged.

The program defaults to line 380, which contains a decision statement.

The first statement in line 380 (CP = PEEK(37) − 1) causes the vertical position of the cursor to be stored one line (− 1) above its current position. The vertical position of the cursor is stored at address location 37 in memory. Line 380 stores the cursor value into the variable, CP, for future use. Later in the program, line 730 returns the cursor position to the CP value, and CALL −958 clears the screen below the cursor, so requests to change the records can be printed on a partially cleared screen.

The second statement in line 380 (IF QU$ = "C" THEN GOSUB 730) is the decision statement that branches to the subroutine at line 730, if changes to the records are to be made.

If in the second statement in line 380, no changes to records are to be made, the program defaults to line 390

```
GOSUB 560
```

The subroutine beginning at line 560 writes the unchanged record to the DUMMY file.

```
580 PRINT D$;"PR#0"
```

CHANGES THE DIRECTION SO THE RECORDS ARE PRINTED TO THE CRT, AND NOT ON THE DISK. The return in line 590 causes the program to jump to line

```
400 GOTO 300
300 GOSUB 520
```

then reads the next record in the file.

If the second statement in line 380 (IF QU$ = "C" THEN GOSUB 730) is TRUE, the program branches to the subroutine beginning at line 730 to begin making the necessary changes to the records.

Line 730 (POKE 37,CP : CALL −958) sets up the vertical cursor position from the value stored at address location 37. CALL −958 clears the

screen below the cursor setting, so the menu generated by lines 730 through 770 can be placed on the CRT. The menu consists of the following items:

```
A TO ALTER
I TO INSERT A NEW RECORD
D TO DELETE
C TO ACCEPT

ENTER CODE?

780 ON ((QU$ = "I")+(QU$ = "A")*2 + (QU$ ="D")*3 + (QU$ = "C")*4) GOTO
790, 980, 1070, 1060 : GOTO 730
```

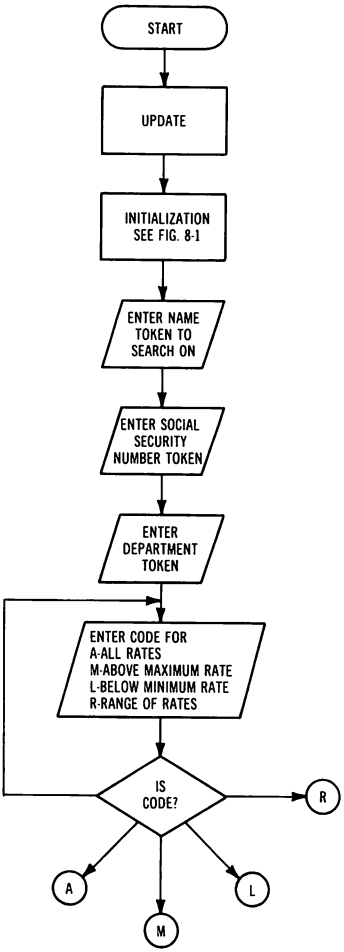


Fig. 8-8. Flowchart of program to read (list) a sequential text file.

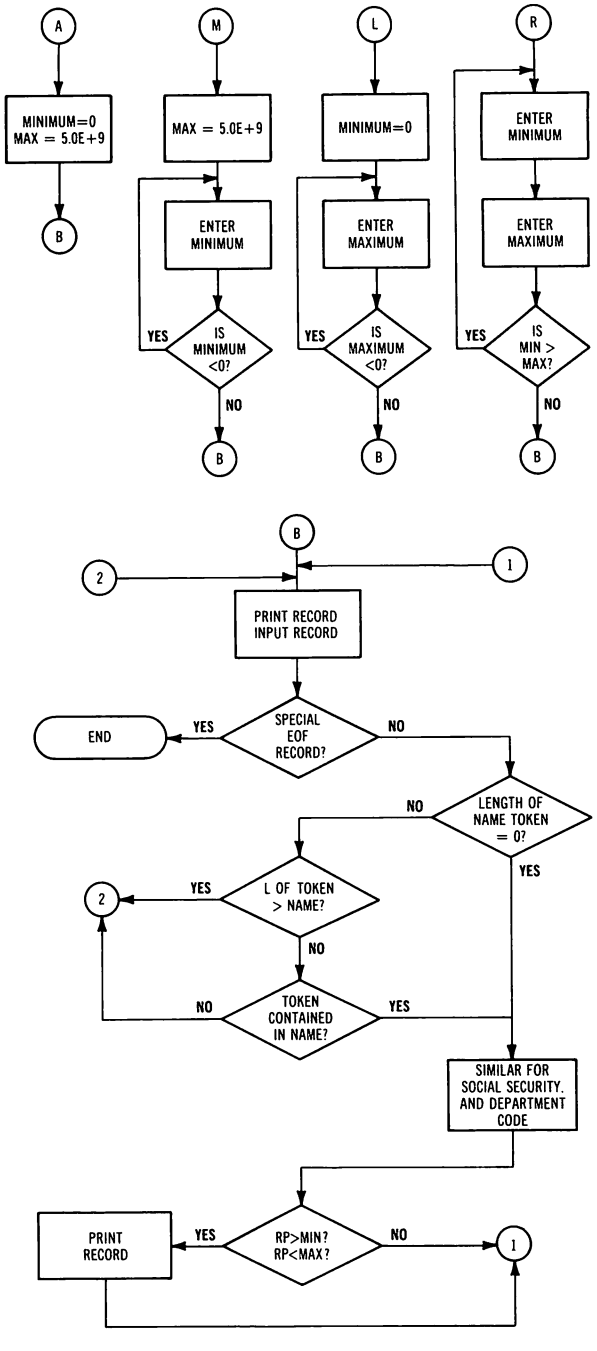


Fig. 8-8—*cont.* Flowchart of program to read (list) a sequential text file.

Line 780, Fig. 8-6, uses an assigned GOTO statement to compute the number of the line to which the program branches. The relationship of the decision is determined by a logical TRUE or FALSE. If the decision is TRUE (1), the 1 is multiplied by the multiplier. The total value, 1 times the multiplier, causes the program to branch to the first, second, third, or fourth line number. In the first logical decision (QU$ = "I"), the multiplier (1) is understood, since TRUE, 1 * 1 = 1, or FALSE, 0 * 1 = 0. In all other logical decisions in the statement, the multiplier is stated.

In line 730, only one logical decision can be true. When one decision is true, all other decisions are false. The false decisions produce a zero value, so zero times the multipliers produces a zero value. The last statement in line 780 is GOTO 730. If a selection is made accidentally, or purposely, that is other than the letters, A, I, D, or C, the input is illegal. The illegal input causes the program to jump back to line 730 for another try at a legal input.

The options to alter the record, insert a new record, delete an item, or accept the record as is will be discussed in reverse order as they appear in the program.

From line 780, if QU$ = "C" is the true decision, or accept the record as it is, the program branches to line 1060. Line

```
1060 RETURN
```

causes the subroutine beginning at line 730 to return to line

```
390 GOSUB 560
```

GOSUB 560 causes the record that was accepted as is to be written to the DUMMY file.

From line 780, if QU$ = "D" is the true decision, the record is deleted from the file. The program branches to line 1070.

```
1070 POP : GOTO 300
```

POP is an Applesoft language command that allows a program to jump out of a subroutine without using the RETURN at the end of the subroutine. The POP causes one RETURN address to pop off the top of the stack that holds the return addresses. POP is used in this program to demonstrate how it is used. POP is not the ideal way to leave a subroutine.

After the instruction POP causes the jump out of the subroutine, the program jumps to line 300.

```
300 GOSUB 520
```

causes a jump to the subroutine that begins at line 520, to cause the program to read the next record in the file.

From line 780, if QU$ = "A" is a true decision, to alter the record, the program branches to line 980.

Line 980 (POKE 37, CP : CALL −958) stores the vertical position of the cursor in address location 37, so the cursor will return to the same position on the CRT each time during the routine. CALL −958 clears the screen below the cursor.

```
980 PRINT "ENTER NEW VALUE OR 'RETURN'"
990 PRINT "TO LEAVE CURRENT VALUE UNCHANGED"
```

Lines 980 and 990 give instructions to the user. If the user wishes to alter the record, the new values are typed in from the keyboard. If the values are to remain the same, RETURN is pressed.

In line 1000, the routine GOSUB 690 places "NAME = " on the screen. The name change is typed in the blanks following the equals sign. The name is placed in QU$. When the program returns from the subroutine to line 1000, IF LEN(QU$) is greater than zero, then QU$ is assigned to NA$. When "NAME = " is placed on the screen, and RETURN is pressed, the name in NA$ is not changed.

Line 1020 (GOSUB 700) places "S SEC NUM = " on the screen. If the social security number is to be altered (LEN(QU$ > 0), the new value is placed in QU$. When the program returns from the subroutine to line 1020, if the length of the new social security number is greater than zero, the value in QU$ is assigned to SS$. If the social security number is not changed, and RETURN was pressed, the social security number is retained in SS$.

Line 1030 (GOSUB 710) places "DEPT = " on the screen. If the department code is altered the value is placed in QU$. When the program returns from the subroutine to line 1030, if the new department code is greater than zero, the value is assigned to DP$. If the department code is not changed, RETURN is pressed, and the department code is retained in DP$.

Line 1040 (GOSUB 720) places "RATE = " on the screen. If the rate of pay is to be altered, the value is placed in QU$. When the program returns from the subroutine to line 1040, if the length of QU$ is greater than zero, the value is assigned to RP = VAL(QU$).

The value in an alphanumeric string variable cannot be used in mathematical computations. The value held in QU$ must be transferred into a numeric variable (RP = VAL(QU$). VAL is the basic word that transfers

the value held in an alphanumeric variable into a numeric variable. If the rate of pay is not altered, RETURN is pressed, and the value originally stored in RP is retained.

```
1050 GOSUB 490 : IF FL THEN PRINT "ILLEGAL RECORD!!!" : FOR J = 1 TO 500 :
NEXT J : PRINT CHR$ (7) : GOTO 980
```

When all alterations have been made to the record, the program defaults to line 1050. The subroutine that begins at line 490 checks the record just input to determine if it is the same as the end of file record. If the input record and the end of file record are the same, the flag value is set to TRUE (1). When the program returns to line 1050, if FL is TRUE (1), the program prints ILLEGAL RECORD!!! on the screen. The pause loop increments 500 times (FOR J = 1 TO 500 : NEXT J), and the computer rings a bell. The program jumps to line 980 to allow the user to input a record that is not the same as the end of file record.

If the program returns from the subroutine that begins at line 490, and the flag value is zero (FALSE), the program defaults to line 1060 to return from the subroutine. The RETURN causes the program to jump to line 390—GOSUB 560. The subroutine that begins at line 560 writes the altered record, or unaltered record, to the DUMMY file.

From line 780, if QU$ = "I" is a true statement, to insert a record into the file, the program branches to line 790.

```
790 L$ = NA$ + SS$ + DP$ + STR$(RP)
800 L1 = LEN(NA$) : L2 = LEN(SS$) : L3 = LEN(DP$)
```

Temporary variables L$, L1, L2, and L3 hold key values of the record presently in memory, that is, the last record read off the input file. The reason to concatenate and store the last record into L$ is that NA$, SS$, DP$, and RP are to be used to input the new inserted record values. Had L$ not been used as a temporary variable, new variables would be needed to input the new input record.

In line 800, the lengths of the values of NA$, SS$, and DP$ are stored in L1, L2, and L3.

The values held in L$, L1, L2, and L3 will be used in lines 890 through 960 to reconstruct the record as it was read from the file, after the insertions have been completed.

Lines 810 through 870 perform a similar function in inputting the record, as lines 980 through 1060 when altering the values stored in the record. Both sets of lines use the lines 690 through 720 to input the individual fields in the record.

```
810 POKE 37,CP : CALL -958
```

stores the vertical cursor value in memory location 37, and CALL −958 clears the screen below the cursor.

```
820 GOSUB 690 : NA$ = QU$
```

GOSUB 690 causes "NAME = " to be printed on the screen. The employee's name is stored in QU$. When the program returns to the second statement in line 690, the value stored in QU$ is assigned to NA$.

```
830 GOSUB 700 : SS$ = QU$
```

GOSUB 700 causes "S SEC NUM = " to be printed on the screen. The employee's social security number is stored in QU$. When the program returns to the second statement on line 830, the social security value stored in QU$ is assigned to SS$.

```
850 GOSUB 720 : RP = VAL(QU$)
```

GOSUB 720 causes "RATE = " to be printed on the screen. The employee's rate of pay is stored in QU$. When the program returns to the second statement in line 850, the alphanumeric string value stored in QU$ is converted to a numeric value and assigned to RP. The basic word VAL converts the alphanumeric string value to a numeric variable.

```
860 GOSUB 490 : IF FL THEN PRINT "THIS IS AN ILLEGAL RECORD!!!" : FOR
J = 1 TO 500 : NEXT J : PRINT CHR$(7) : GOTO 810
```

GOSUB 490 causes the program to jump to the subroutine that begins at line 490 to check the record to be inserted into the file, against the end of file record. If the two records are the same, the flag value is set to TRUE (1). The program returns to line 860 to test the flag value. The flag value is TRUE (1), so the program prints out that the record just input is an illegal record. The pause loop increments for 500 counts (FOR J = 1 TO 500 : NEXT J), and the computer bell rings (PRINT CHR$(7)), and the program jumps to line 810, so a legal record can be input.

When the program returns from the subroutine that begins at line 490, if the flag value is zero (FALSE), the program returns to line 870.

```
870 GOSUB 560 : INPUT "INSERT ANOTHER RECORD (Y OR N)?";QU$
```

The program jumps to the subroutine that begins at line 560 to write the record to the DUMMY file. The program returns to the second statement in line 870.

```
880 IF QU$ = "Y" THEN 810
```

If QU$ = "Y", and the user is to input another record, the program jumps to line 810.

If QU$ = a value other than "Y", the program defaults to line 890, to reconstruct the file record that was temporarily stored in L$, L1, L2, and L3.

```
890 L0 = LEN(L$)
```

The value of the length of L$ is assigned to the variable L0.

```
900 NA$ = LEFT$(L$,L1)
```

The LEFT$ function is used to break the employee's name out of the temporary storage variable L$. The length of the employee's name is stored in the variable L1.

```
910 L$ = RIGHT$(L$,L0-L1) : L0 = L0-L1
```

After the value of the employee's name has been taken out of L$, the remaining values of the three other fields in the record are assigned to the temporary L$ storage location. The value of the length of the employee's name is subtracted from the total value of the length of the string array – L0 = L0–L1.

```
920 SS$ = LEFT$(L$,L2)
```

After the employee's name is removed from the temporary storage variable L$, the SS$ occupies the first position in L$. In line 920 the social security number is assigned to SS$.

```
930 L$ = RIGHT$(L$,L0-L2) : L0 = L0-L2
```

The remaining two fields of the record are now assigned to the temporary storage location L$. The value of the length of the social security number is subtracted from the total length of the variable L0 = L0–L2. There are now two fields stored in L$, the DP$, and the RP. In line 940 the DP$ is assigned the value in the department code string.

```
950 L$ = RIGHT$(L$,L0-L3) : L0 = L0-L3
```

The remaining field RP is the only part of the record that remains in the original temporary L$ variable. In line 960, the RP field is assigned to the RP through the use of the VAL function.

```
960 RP = VAL(L$)
```

Thus, the procedure is complete. The record was concatenated in line 790, and the complete record was stored in the temporary variable L$. The

```
100  REM :PROGRAM TO READ A       SEQUENTIAL FILE
110 D$ =  CHR$ (4)
120 SL = 6: REM :SLOT IS #6
130 DR = 1: REM :DRIVE IS #1
140  HOME : VTAB 3
150  PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
160  PRINT
170  INPUT "CHANGE THESE VALUES(Y OR N)? ";QU$
180  PRINT
190  IF QU$ <  > "Y" THEN 240
200  INPUT "ENTER SLOT (1-7) ?";SL
210  IF SL < 1 OR SL > 7 THEN 200
220  INPUT "ENTER DRIVE (1-2) ?";DR
230  IF DR <  > 1 AND DR <  > 2 THEN 220
240  PRINT : INPUT " ENTER FILE NAME ?";F$
250  IF  LEN (F$) = 0 THEN  PRINT : PRINT "*** NAME IS
      INVALID ***": GOTO 240
260  FOR J = 1 TO  LEN (F$): IF  ASC ( MID$ (F$,J,1)) <
     32 THEN  PRINT : PRINT "*** PLEASE DON'T TRY CONT
     ROL CHARA-": PRINT "CTERS YET !": GOTO 240
270  NEXT J: PRINT : PRINT "!!! THE NAME IS GOOD"
280  PRINT D$;"OPEN ";F$;",S";SL;",D";DR
290  HOME : PRINT "ENTER VALUE FOR SELECTION ELSE"
300  PRINT : PRINT "PRESS 'RETURN' TO LIST ALL ENTRIES
     "
310  PRINT : INPUT "ENTER NAME TOKEN ?";TN$
320  PRINT : INPUT "ENTER SS NUM TOKEN ?";TS$
330  PRINT : INPUT "ENTER DEPT TOKEN ?";DT$
340  HOME : PRINT : PRINT "ENTER RANGE OF PAY RATE<R>"

350  PRINT : PRINT "SELECT ABOVE MINIMUM <M>"
360  PRINT : PRINT "SELECT BELOW MAXIMUM<L>"
370  PRINT : PRINT "ACCEPT ALL RATES<A>"
380  PRINT : INPUT "ENTER SELECTION ?";QU$
390  ON ((QU$ = "M") + (QU$ = "L") * 2 + (QU$ = "R") *
     3 + (QU$ = "A") * 4) GOTO 400,430,460,490: GOTO 3
     40
400  HOME : PRINT : INPUT "ENTER MINIMUM RATE ?";LR
410 UR = 5.0E + 9: IF LR < 0 THEN 400
420  GOTO 500
430  HOME : PRINT : INPUT "ENTER UPPER LIMIT ?";UR
440 LR = 0: IF LR > UR THEN 430
450  GOTO 500
460  HOME : INPUT "ENTER BASE FIGURE ?";LR: INPUT "ENT
     ER MAXIMUM FIGURE ?";UR
470  IF LR < 0 OR UR < 0 OR UR < LR THEN 460
480  GOTO 500
490 LR = 0:UR = 5E + 9
500  HOME
510  PRINT D$;"READ ";F$
520  INPUT NA$,SS$,DP$,RP
530  IF NA$ = "END RECORD" AND SS$ = "000-00-0000" AND
     DP$ = "00" AND RP = 0 THEN 760
540 L =  LEN (TN$): IF L = 0 THEN 590
550  IF  LEN (NA$) < L THEN 750
560  FOR J = 1 TO  LEN (NA$) - L + 1
```

Fig. 8-9. Program to read (list) all, or parts of, a sequential text file.

lengths of the first three fields were stored in temporary variables in line 800.

In lines 890 through 960 the process was reversed. The temporary variable L$ was broken down, field by field, and the record was stored in its original variables.

Line 970 (GOTO 730) causes the program to jump to line 730 to begin the input selection again.

READ (LIST) A SEQUENTIAL TEXT FILE

The third program, Fig. 8-9, allows the user to read all of a sequential text file, and list all, or parts, of the file. The program has two prime purposes, (1) to list all the records in the file TEST-RECORDS, or (2) to search through the file TEST-RECORDS for specific records.

The program offers 32 options on which to list and/or search. There are two possibilities per token: (1) RETURN is pressed and a null value is entered, which will allow the program to accept that field of the record, or (2) one or more characters are entered, which will cause the program to check to see if the characters in the token are contained in the equivalent record field. There are four options available when searching on the pay rate: (1) all pay rates, (2) a minimum and maximum range, (3) pay rates below a maximum rate, and (4) pay rates above a minimum pay rate. A search can be conducted using any combination of the 32 options.

```
570  IF TN$ =  MID$ (NA$,J,L) THEN 590
580  NEXT J: GOTO 750
590 L =  LEN (TS$): IF L = 0 THEN 640
600  IF  LEN (SS$) < L THEN 750
610  FOR J = 1 TO  LEN (SS$) - L + 1
620  IF TS$ =  MID$ (SS$,J,L) THEN 640
630  NEXT J: GOTO 750
640 L =  LEN (DT$): IF L = 0 THEN 690
650  IF  LEN (DP$) < L THEN 750
660  FOR J = 1 TO  LEN (DP$) - L + 1
670  IF DT$ =  MID$ (DP$,J,L) THEN 690
680  NEXT J: GOTO 750
690  IF RP < LR OR RP > UR THEN 750
700  PRINT "NAME =     ";NA$
710  PRINT "S SEC # = ";SS$
720  PRINT "DEPT     = ";DP$
730  PRINT "PAY RATE= ";RP
740  PRINT : PRINT "----------": PRINT
750  GOTO 510
760  PRINT D$;"IN#0"
770  END
```

Fig. 8-9—*cont.* Program to read (list) all, or parts of, a sequential text file.

Fig. 8-10A is a search conducted using "ZEB" as a name token. Fig. 8-10B is a search using "111" as a social security token. Fig. 8-10C is a search using the letter "B" as a department code token (no record was found because "B" was not in any department code). Fig. 8-10D is a search for all employees whose pay range is between $8.00 and $10.00 per hour. Fig. 8-10E is a search for all employees whose pay is above a minimum of $5.00 per hour. Fig. 8-10F is a search for all employees whose pay is below a $9.00 maximum per hour. Fig. 8-10G is a list of all employees in the file TEST-RECORDS.

Lines 110 through 280, of the program in Fig. 8-9, are identical to the same lines in the first two programs, Figs. 8-2 and 8-5. Lines 290 through 370 are interactive questions to determine on what basis the search is to be conducted.

```
290 HOME : PRINT "ENTER VALUE FOR SELECTION ELSE"
300 PRINT : PRINT "PRESS 'RETURN' TO LIST ALL ENTRIES"
310 PRINT : INPUT "ENTER NAME TOKEN ?";TN$
320 PRINT : INPUT "ENTER SS NUM TOKEN ?";TS$
330 PRINT : INPUT "ENTER DEPT TOKEN ?";DT$
```

```
]RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N


 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
ENTER VALUE FOR SELECTION ELSE

PRESS 'RETURN' TO LIST ALL ENTRIES

ENTER NAME TOKEN ?ZEB

ENTER SS NUM TOKEN ?

ENTER DEPT TOKEN ?

ENTER RANGE OF PAY RATE<R>

SELECT ABOVE MINIMUM <M>

SELECT BELOW MAXIMUM<L>

ACCEPT ALL RATES<A>

ENTER SELECTION ?A
NAME =     HELEN ZEBRONSKI
S SEC # = 000-000-0000
DEPT     = D8
PAY RATE= 10.5

----------
```

(A) Search using name token ZEB.

Fig. 8-10. Search of program in Fig. 8-9.

```
 RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N

 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
ENTER VALUE FOR SELECTION ELSE

PRESS 'RETURN' TO LIST ALL ENTRIES

ENTER NAME TOKEN ?

ENTER SS NUM TOKEN ?111

ENTER DEPT TOKEN ?

ENTER RANGE OF PAY RATE<R>

SELECT ABOVE MINIMUM <M>

SELECT BELOW MAXIMUM<L>

ACCEPT ALL RATES<A>

ENTER SELECTION ?A
NAME =     SUE ADAMS
S SEC # = 111-11-1111
DEPT    = C3
PAY RATE= 8.5

----------
```

(B) Search using the social security token 111.

```
 RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N

 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
ENTER VALUE FOR SELECTION ELSE

PRESS 'RETURN' TO LIST ALL ENTRIES

ENTER NAME TOKEN ?

ENTER SS NUM TOKEN ?

ENTER DEPT TOKEN ?B

ENTER RANGE OF PAY RATE<R>

SELECT ABOVE MINIMUM <M>

SELECT BELOW MAXIMUM<L>

ACCEPT ALL RATES<A>

ENTER SELECTION ?A
```

(C) Search using a department code token B.

Fig. 8-10—*cont.* Search of program in Fig. 8-9.

```
 RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N


 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
ENTER VALUE FOR SELECTION ELSE

PRESS 'RETURN' TO LIST ALL ENTRIES

ENTER NAME TOKEN ?

ENTER SS NUM TOKEN ?

ENTER DEPT TOKEN ?

ENTER RANGE OF PAY RATE<R>

SELECT ABOVE MINIMUM <M>

SELECT BELOW MAXIMUM<L>

ACCEPT ALL RATES<A>

ENTER SELECTION ?R
ENTER BASE FIGURE ?8.00
ENTER MAXIMUM FIGURE ?10.00
NAME =     JOHN HO
S SEC # = 000-00-0000
DEPT     = C6
PAY RATE= 9.6
----------
NAME =     SUE ADAMS
S SEC # = 111-11-1111
DEPT     = C3
PAY RATE= 8.5
----------
NAME =     JIM BELL
S SEC # = 222-22-2222
DEPT     = A5
PAY RATE= 8.25

----------
```

(D) Search using a range of pay rates.

Fig. 8-10—*cont.* Search of program in Fig. 8-9.

```
 RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N


 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
ENTER VALUE FOR SELECTION ELSE

PRESS 'RETURN' TO LIST ALL ENTRIES

ENTER NAME TOKEN ?

ENTER SS NUM TOKEN ?

ENTER DEPT TOKEN ?

ENTER RANGE OF PAY RATE<R>

SELECT ABOVE MINIMUM <M>

SELECT BELOW MAXIMUM<L>

ACCEPT ALL RATES<A>

ENTER SELECTION ?M

ENTER MINIMUM RATE ?5.00
NAME =     JOHN HO
S SEC # = 000-00-0000
DEPT     = C6
PAY RATE= 9.6

----------

NAME =     HELEN ZEBRONSKI
S SEC # = 000-00-0000
DEPT     = D8
PAY RATE= 10.5

----------

NAME =     SUE ADAMS
S SEC # = 111-11-1111
DEPT     = C3
PAY RATE= 8.5

----------

NAME =     JIM BELL
S SEC # = 222-22-2222
DEPT     = A5
PAY RATE= 8.25

----------
```

(E) Search using above a minimum limit of $5.00 of pay rates.

Fig. 8-10—*cont.* Search of program in Fig. 8-9.

```
 RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N


 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
ENTER VALUE FOR SELECTION ELSE

PRESS 'RETURN' TO LIST ALL ENTRIES

ENTER NAME TOKEN ?

ENTER SS NUM TOKEN ?

ENTER DEPT TOKEN ?

ENTER RANGE OF PAY RATE<R>

SELECT ABOVE MINIMUM <M>

SELECT BELOW MAXIMUM<L>

ACCEPT ALL RATES<A>

ENTER SELECTION ?L

ENTER UPPER LIMIT ?9.00
NAME =     SUE ADAMS
S SEC # = 111-11-1111
DEPT     = C3
PAY RATE= 8.5

----------

NAME =     JIM BELL
S SEC # = 222-22-2222
DEPT     = A5
PAY RATE= 8.25

----------
```

(F) Search using below a maximum limit of $9.00 of pay rates.

Fig. 8-10—*cont.* Search of program in Fig. 8-9.

```
 RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N


 ENTER FILE NAME ?TEST-RECORDS

!!! THE NAME IS GOOD
ENTER VALUE FOR SELECTION ELSE

PRESS 'RETURN' TO LIST ALL ENTRIES

ENTER NAME TOKEN ?

ENTER SS NUM TOKEN ?

ENTER DEPT TOKEN ?

ENTER RANGE OF PAY RATE<R>

SELECT ABOVE MINIMUM <M>

SELECT BELOW MAXIMUM<L>

ACCEPT ALL RATES<A>

ENTER SELECTION ?A
NAME =     JOHN HO
S SEC # = 000-00-0000
DEPT     = C6
PAY RATE= 9.6

----------

NAME =     HELEN ZEBRONSKI
S SEC # = 000-00-0000
DEPT     = D8
PAY RATE= 10.5

----------

NAME =     SUE ADAMS
S SEC # = 111-11-1111
DEPT     = C3
PAY RATE= 8.5

----------

NAME =     JIM BELL
S SEC # = 222-22-2222
DEPT     = A5
PAY RATE= 8.25

----------


]PR#0
```

(G) Lists all the records in the TEST-RECORDS file.

Fig. 8-10—*cont.* Search of program in Fig. 8-9.

```
340 HOME : PRINT : PRINT "ENTER RANGE OF PAY RATE<R>"
350 PRINT : PRINT "SELECT ABOVE MINIMUM <M>"
360 PRINT : PRINT "SELECT BELOW MAXIMUM <L>"
370 PRINT : PRINT "ACCEPT ALL RATES<A>"
380 PRINT : INPUT "ENTER SELECTION ?";QU$
```

The first RUN of the program in Fig. 8-9 is a search, Fig. 8-10A, of the records in TEST-RECORDS, on the name token ZEB (ZEBRONSKI). After the program verifies the file name is good, in lines 260 and 270, the program opens the file, TEST-RECORDS, line 280. The program defaults through lines 290 and 300, and stops at line 310 to ask if a specific name or name token is to be the object of the search. In this example, ZEB is the name token that is the object of the search. When the name token is contained in the employee's name field, all four fields of the record are printed. RETURN is pressed for each line, 320 through 370, until line 380 comes on the screen. When RETURN is pressed, a null value is placed in the related variable.

```
380 PRINT : INPUT "ENTER SELECTION ?";QU$
```

Line 390 is similar to the explanation in Fig. 8-6, and the logic is the same.

Inputting the name token ZEB and "A", to list all records with the ZEB token, causes the program to branch to line 490.

```
490 LR = 0 : UR = 5E+9
```

Line 490 sets the upper and lower ranges for the pay scale. This has no relation to the name search, but must be set for searches on the pay rate that also use lines 500 through 510.

The program defaults to line 500 (HOME) and clears the screen. Lines 510 and 520 read a record from the file. Line 530 checks to determine if the record just read from the file is the end of file record. If the end of file record is read, the program branches to line 760, to return input control to the keyboard.

```
760 PRINT D$;"IN#0"
```

```
510 PRINT D$;"READ";F$
520 INPUT NA$; SS$; DP$; RP
530 IF NA$ = "END RECORD" AND SS$ = "000-00-0000" AND DP$ = "00" AND
    RP = 0 THEN 760
```

If line 530 is logically FALSE, and the end of file record has not been read, the program defaults to line 540 to store the length of the name token

(TN$) in the variable "L." If L = 0 (RETURN was pressed in line 310), the program branches to check the social security token TS$, at line 590. If L = 0, the program accepts this record based on the employee name in the record.

```
540 L = LEN(TN$) : IF L = 0 THEN 590
550 IF LEN(NA$) < THEN 750
```

In line 550 if the length of NA$, the name in the record just read, is less than the length of the name token (L of TN$), then the record cannot be checked because the name token is longer than the name read from the record. The program branches to line 750 (GOTO 510), so the user can read the next record in the file to check against the name token.

If the name token is less than the name read from the record, line 550 is logically FALSE, the program defaults to line 560 to compare the individual characters in the name token, and the name in the record.

```
560 FOR J = 1 TO LEN(NA$)-L+1
570 IF TN$ = MID$(NA$,J,L) THEN 590
580 NEXT J : GOTO 750
```

Lines 560, 570, and 580 set up the loop and the decision statement to determine whether the characters in the token are in the name contained in the record, Fig. 8-11. Each loop pass checks three characters in the name string with the name token, until the name token is the same as the name string, or the end of the name string is reached, and no equality is found.

```
L=LEN(TN$) OF ZEB=3    LEN(NA$) OF BENBOW=6
560 FOR J=1 TO LEN(NA$)-L+1
570 IF TN$=MID$(NA$,J,L) THEN 590
580 NEXT J: GOTO 750

FOR J=1 TO LEN(NA$)-L+1
                    6-3+1=
                    6-3=3+1=4

FOR J=1 TO 4
         TN$= MID$(NA$), J, L
         ZEB=       BEN    1   3
         ZEB=       ENB    2   3
         ZEB=       NBE    3   3
         ZEB=       BOW    4   3

NEXT J: GOTO 750

(NO MATCH WAS FOUND)
```

Fig. 8-11. Using the name token ZEB to search a record with the name BENBOW in the employee name field.

If no equality is found, the loop terminates, and the program jumps to line 750,

```
750 GOTO 510
```

and another record is read from the file to compare to the name token. If an equality is found between the name token and the name from the record, line 570 jumps out of the loop to line 590. There were no other tokens input, so the program defaults to line 700. Each token input is checked with the equivalent employee record until a match is found. If a match is found, the employee record is printed, lines 700 through 740. If no match is found, the search continues to the end of the file. When the search is complete, the program ends.

The second RUN of the program in Fig. 8-9, is a search (Fig. 8-10B) for a social security token. When line 310 (PRINT : INPUT "ENTER NAME TOKEN ?";TN$) was reached, RETURN was pressed, which placed a null value in TN$. The program defaults to line

```
320 PRINT : INPUT "ENTER SS NUM TOKEN ?";TS$
```

In Fig. 8-10B, "111" was input as the social security number token. RETURN was pressed for line 330, and lines 340, 350, 360, and 370 are printed on the screen.

In line 380 "A", for all PAY RATES, was input which causes the program to branch to line 490. Line 490 sets the lower and upper pay ranges. Line 500 clears the screen. Since no name token was input in line 310, the name token value is null. A null string value equates to a variable, "L", value of zero. Line 540 IF L = 0 THEN 590 causes the program to branch to line 590.

```
590 L = LEN(TS$) : IF L = 0 THEN 640
600 IF LEN(SS$) < L THEN 750
610 FOR J = 1 TO LEN(SS$)-L+1
620 IF TS$ = MID$(SS$,J,L) THEN 640
630 NEXT J : GOTO 750
```

In line 590, the value of the length of the social security token is assigned to the variable "L". If L = 0, no social security token was input in line 320, and RETURN was pressed.

In line 320, the social security token input was "111", so the value stored in the variable "L" is three (3).

In line 600, the length of the social security number (000–00–0000) is 11, so the social security value is greater than the social security token (3). The loop, lines 610 through 630, is similar to the structure in Fig. 8-11.

The loop increments seven times (LEN(SS$)(11) − L + 1(4)), and compares the three characters in the social security string to the social security token. If a match is found, the record is printed out using lines 700 through 740. If no match is found, the program jumps to line 750,

```
750 GOTO 510
```

and then to line 510, to read in another record. This continues until the end of the file is reached. When the search is complete, the program ends.

The third RUN of the program in Fig. 8-9 is a search (Fig. 8-10C) for the employees with a department code token of "B". In this case, there is no department code that has a "B" in its code designation, so no record is printed out.

In this RUN, no name token is input and no social security token is input. The department code token causes the lines from 640 through 680 to compare the department code token to the department codes in all the records on the file. Lines 640 through 680 are used in a similar fashion as the explanation in Fig. 8-11. The program ended with no record printed out.

The fourth RUN of the program is to search for those employees whose pay rate is between $8.00 per hour and $8.60 per hour (Fig. 8-10D). RETURN is pressed for each request for a token input until line 380.

```
340 HOME : PRINT : PRINT "ENTER RANGE OF PAY RATE <R>"
```

When "R" is typed in response to line 380 ("ENTER SELECTION ";QU$), the program branches to line 460.

```
460 HOME : INPUT "ENTER BASE FIGURE ?";LR : INPUT "ENTER MAXIMUM
  FIGURE ?";UR
470 IF LR < 0 OR UR < 0 OR UR < LR THEN 460
480 GOTO 500
```

Line 460 asks the user to input the lower range of the pay rate, and the upper range of the pay rate. The search is to be conducted for those employees who make the minimum and above, and those employees who make the maximum and below. Since RETURN was pressed for all tokens to line 380, the null values cause the program to default lines 500 through 680.

```
690 IF RP < LR OR RP > UR THEN 750
```

Line 690 eliminates the records where the pay rate is less then the low range (minimum), and the pay rate is greater than the upper range (maximum). Those employee records that come within the minimum and maximum pay rates are printed out in lines 700 through 740. Line 750 (GOTO 510) causes the program to jump to line 510 to read the next record in the

file. When the last record in the file is read, the program branches to line 760 to return input control to the keyboard, and the program ends at line 770.

FOR THE INTEGER BASIC USER

Presented in Figs. 8-12, 8-13, and 8-14 are the equivalent CREATE, UPDATE, and READ programs for the Integer BASIC user. The line numbering remains the same as far as possible. Several lines had to be inserted because of the different way in which Applesoft and Integer BASIC handle the statements following the IF-THEN statement.

For example, look at lines 1050 and 1055 in Fig. 8-13 and compare these to line 1050 in Fig. 8-5. The logic has to be reversed and the new line performs all the functions that followed the THEN in Applesoft. Of course, various Applesoft features that are not in Integer BASIC must somehow be

```
 10 DIM CHR$(27)
 20 FOR J=1 TO 26: POKE 2054+J,J+128: NEXT J
 30 POKE 2081,30
100 REM :PROGRAM TO CREATE A         SEQUENTIAL FILE
110 D$=CHR$(4,4)
115 DIM F$(30)
120 SL=6: REM :SLOT # IS 6
130 DR=1: REM :DRIVE # IS 1
140 CALL -936: VTAB 3
150 PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
160 PRINT
170 INPUT "CHANGE THESE VALUES(Y OR N)? ",QU$
180 PRINT
190 IF QU$#"Y" THEN 240
200 INPUT "ENTER SLOT (1-7) ",SL
210 IF SL<1 OR SL>7 THEN 200
220 INPUT "ENTER DRIVE (1-2) ",DR
230 IF DR<1 OR DR>2 THEN 220
240 PRINT : INPUT "ENTER FILE NAME? ",F$
250 IF LEN(F$)>0 THEN 260
255 PRINT : PRINT "*** NAME IS INVALID ***": GOTO 240

260 FOR J=1 TO LEN(F$): IF ASC(F$(J,J))< ASC(CHR$(1,
    1)) OR ASC(F$(J,J))> ASC(CHR$(26,26)) THEN 270
265 PRINT "PLEASE DON'T TRY CONTROL CHARACTERS ": FOR
    J=1 TO 500: NEXT J: GOTO 240
270 NEXT J: PRINT "THE NAME IS GOOD!!!"
280 PRINT D$;"OPEN ";F$;",S";SL;",D";DR
290 PRINT D$;"WRITE ";F$
300 PRINT "END RECORD": PRINT "000-00-0000": PRINT "00"
    : PRINT "00.00"
310 PRINT D$;"CLOSE"
320 PRINT D$;"LOCK ";F$;",S";SL;",D";DR
330 END
```

Fig. 8-12. Program to create a sequential text file in Integer BASIC.

```
10 DIM CHR$(27)
20 FOR J=1 TO 27: POKE 2054+J,J+128: NEXT J
30 POKE 2081,30
40 GOTO 100
50 CI= PEEK (37):CH= PEEK (36)
51 POKE 37,CI: POKE 36,CH: CALL -958: INPUT " ",NUM$

52 L= LEN(NUM$): IF L=0 THEN GOTO 51
53 D=0
54 FOR I=1 TO L
55 IF NUM$(I,I)=" " THEN GOTO 51
56 IF NUM$(I,I)#"." THEN GOTO 58
57 IF D#0 THEN GOTO 51:D=I: GOTO 59
58 IF ASC(NUM$(I,I))<176 OR ASC(NUM$(I,I))>185 THEN
   GOTO 51
59 NEXT I: IF (D=0 AND L>8) OR (D#0 AND L-D>2) THEN
   51: GOTO 61
60 L= LEN(NUM$):D=L-2
61 CNTS=0:BDOL=0:SDOL=0:DIS$="............."
62 J=D-1: IF D=0 THEN J=L
63 K=12-J-(J-1)/3:DIS$(K-1)="$": IF D=1 THEN 70
64 FOR I=1 TO J
65 IF K MOD 4>0 THEN 67
66 DIS$(K)=",":K=K+1
67 DIS$(K)=NUM$(I,I): IF (J-I)/4 THEN 68:SDOL=SDOL*
   10+ ASC(NUM$(I,I))-176: GOTO 69
68 BDOL=BDOL*10+ ASC(NUM$(I,I))-176
69 K=K+1: NEXT I
70 DIS$(12)="."
71 IF D>0 AND D<L THEN 72:DIS$(13)="00": GOTO 74
72 IF L-D<2 THEN 73:DIS$(13)=NUM$(D+1):CNTS=( ASC(NUM$
   (D+1,D+1))-176)*10+ ASC(NUM$(D+2,D+2))-176: GOTO
   74
73 DIS$(13)=NUM$(D+1):DIS$(14)="0":CNTS=( ASC(NUM$(
   D+1,D+1))-176)*10
74 RETURN
75 IF VAL#1 THEN 76:AP$="1": RETURN
76 IF VAL#2 THEN 77:AP$="2": RETURN
77 IF VAL#3 THEN 78:AP$="3": RETURN
78 IF VAL#4 THEN 79:AP$="4": RETURN
79 IF VAL#5 THEN 80:AP$="5": RETURN
80 IF VAL#6 THEN 81:AP$="6": RETURN
81 IF VAL#7 THEN 82:AP$="7": RETURN
82 IF VAL#8 THEN 83:AP$="8": RETURN
83 IF VAL#9 THEN 84:AP$="9": RETURN
84 AP$="0": RETURN
85 NUM$="":TVAL=BDOL:J=0: IF BDOL=0 THEN 89
86 FOR I=1 TO 4
87 VAL=TVAL/10 ^ (4-I): GOSUB 75
88 NUM$(I)=AP$:TVAL=TVAL-VAL*10 ^ (4-I): NEXT I:J=4

89 TVAL=SDOL: IF SDOL=0 AND J=0 THEN 94: IF SDOL>0 THEN
   90:NUM$(J+1)="0000": GOTO 93
90 FOR I=1 TO 4
91 VAL=TVAL/10 ^ (4-I): GOSUB 75
92 NUM$(I+J)=AP$:TVAL=TVAL-VAL*10 ^ (4-I): NEXT I
```

Fig. 8-13. Program to update a sequential text file in Integer BASIC.

```
 93 J=J+4
 94 NUM$(J+1)=".":TVAL=CNTS: IF CNTS>0 THEN 95:NUM$(
    J+2)="00": GOTO 97
 95 FOR I=1 TO 2:VAL=TVAL/10 ^ (2-I): GOSUB 75
 96 NUM$(I+J+1)=AP$:TVAL=TVAL-VAL*10 ^ (2-I): NEXT I
 97 J= LEN(NUM$): FOR I=1 TO J: IF NUM$(I,I)#"0" AND
    NUM$(I,I)#"." THEN 98: NEXT I: RETURN
 98 IF I>1 THEN NUM$=NUM$(I,J): RETURN
100 REM :PROGRAM TO UPDATE A         SEQUENTIAL FILE
110 D$=CHR$(4,4)
115 DIM NUM$(11),DIS$(14),F$(30),NA$(200),SS$(12),DP$
    (20),RP$(20),L$(255),QU$(255)
120 SL=6: REM :SLOT # IS 6
130 DR=1: REM :DRIVE # IS 1
140 CALL -936: VTAB 3
150 PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
160 PRINT
170 INPUT "CHANGE THESE VALUES(Y OR N)? ",QU$
180 PRINT
190 IF QU$#"Y" THEN 240
200 INPUT "ENTER SLOT (1-7) ",SL
210 IF SL<1 OR SL>7 THEN 200
220 INPUT "ENTER DRIVE (1-2) ",DR
230 IF DR<1 OR DR>2 THEN 220
240 PRINT : INPUT "ENTER FILE NAME? ",F$
250 IF LEN(F$)>0 THEN 260
255 PRINT : PRINT "*** NAME IS INVALID ***": GOTO 240

260 FOR J=1 TO LEN(F$): IF ASC(F$(J,J))< ASC(CHR$(1,
    1)) OR ASC(F$(J,J))> ASC(CHR$(26,26)) THEN 270
265 PRINT "PLEASE DON'T TRY CONTROL CHARACTERS ": FOR
    J=1 TO 500: NEXT J: GOTO 240
270 NEXT J: PRINT "THE NAME IS GOOD!!!"
280 PRINT D$;"OPEN ";F$;",S";SL;",D";DR
290 PRINT D$;"OPEN DUMMY,S";SL;",D";DR
300 GOSUB 520
310 IF FL THEN 410
320 CALL -936
330 PRINT "NAME = ";NA$
340 PRINT : PRINT "S SEC NUM =";SS$
350 PRINT : PRINT "DEPT = ";DP$
360 PRINT : PRINT "PAY RATE = ";:NUM$=RP$: GOSUB 60:
     PRINT DIS$
370 PRINT : INPUT "TO CHANGE ENTER <C> ",QU$
380 CP= PEEK (37): IF QU$="C" THEN GOSUB 730
390 GOSUB 560
400 GOTO 300
410 GOSUB 600
420 NA$="END RECORD":SS$="000-00-0000":DP$="00":RP$=
    "00.00": GOSUB 560
430 PRINT D$;"CLOSE"
440 PRINT D$;"UNLOCK ";F$;",S";SL;",D";DR
450 PRINT D$;"DELETE ";F$;",S";SL;",D";DR
460 PRINT D$;"RENAME DUMMY,";F$
470 PRINT D$;"LOCK ";F$;",S";SL;",D";DR
480 END
```

Fig. 8-13—*cont.* Program to update a sequential text file in Integer BASIC.

```
490 FL=0
500 IF NA$="END RECORD" AND SS$="000-00-0000" AND DP$
    ="00" AND RP$="00.00" THEN FL=1
510 RETURN
520 PRINT D$;"READ ";F$
530 INPUT NA$,SS$,DP$,RP$
540 PRINT D$;"IN#0": GOSUB 490
550 RETURN
560 PRINT D$;"WRITE DUMMY"
570 PRINT NA$: PRINT SS$: PRINT DP$: PRINT RP$
580 PRINT D$;"PR#0"
590 RETURN
600 INPUT "DO YOU WANT TO ADD MORE RECORDS? ",QU$
610 IF QU$#"Y" THEN RETURN
620 CALL -936
630 GOSUB 690:NA$=QU$
640 GOSUB 700:SS$=QU$
650 GOSUB 710:DP$=QU$
660 GOSUB 720: GOSUB 85:RP$=NUM$
670 GOSUB 490: IF NOT FL THEN 680
675 PRINT "THIS IS AN ILLEGAL RECORD !!!": FOR J=1 TO
    500: NEXT J: PRINT CHR$(7,7): GOTO 620
680 GOSUB 560: GOTO 600
690 INPUT "NAME = ",QU$: RETURN
700 INPUT "S SEC NUM =",QU$: RETURN
710 INPUT "DEPT = ",QU$: RETURN
720 PRINT "RATE = ";: GOSUB 50: RETURN
730 POKE 37,CP: CALL -958: PRINT "ENTER:": PRINT : TAB
    7: PRINT "A TO ALTER"
740 PRINT : TAB 7: PRINT "I TO INSERT NEW RECORD"
750 PRINT : TAB 7: PRINT "D TO DELETE ITEM"
760 PRINT : TAB 7: PRINT "C TO ACCEPT ITEM AS IS"
770 PRINT : INPUT "ENTER CODE? ",QU$
780 IF QU$="I" THEN 790
782 IF QU$="A" THEN 980
784 IF QU$="D" THEN 1070
786 IF QU$="C" THEN 1060
788 GOTO 730
790 L1= LEN(NA$):L2= LEN(SS$):L3= LEN(DP$)
800 L$=NA$:L0=L1+1:L$(L0)=SS$:L0=L0+L2:L$(L0)=DP$:L0=
    L0+L3:L$(L0)=RP$
810 POKE 37,CP: CALL -958
820 GOSUB 690:NA$=QU$
830 GOSUB 700:SS$=QU$
840 GOSUB 710:DP$=QU$
850 GOSUB 720: GOSUB 85:RP$=NUM$
860 GOSUB 490: IF NOT FL THEN 870
865 PRINT "THIS IS AN ILLEGAL RECORD !!!": FOR J=1 TO
    500: NEXT J: PRINT CHR$(7): GOTO 810
870 GOSUB 560: INPUT "INSERT ANOTHER RECORD (Y OR N )? "
    ,QU$
880 IF QU$="Y" THEN 810
890 L0=1:NA$=L$(L0,L1)
900 L0=L0+L1
910 SS$=L$(L0,L0+L2-1)
920 L0=L0+L2
```

Fig. 8-13—*cont.* Program to update a sequential text file in Integer BASIC.

```
 930 DP$=L$(LO,LO+L3-1)
 940 LO=LO+L3
 950 RP$=L$(LO)
 970 GOTO 730
 980 POKE 37,CP: CALL -958: PRINT "ENTER NEW VALUE OR 'RET
     URN'"
 990 PRINT "TO LEAVE CURRENT VALUE UNCHANGED"
1000 PRINT : GOSUB 690: IF LEN(QU$)>0 THEN NA$=QU$
1020 GOSUB 700: IF LEN(QU$)>0 THEN SS$=QU$
1030 GOSUB 710: IF LEN(QU$)>0 THEN DP$=QU$
1040 CI= PEEK (37):CH= PEEK (36)+7: INPUT "RATE = ",QU$
     : IF LEN(QU$)=0 THEN 1050
1045 NUM$=QU$: GOSUB 52: GOSUB 85:RP$=NUM$
1050 GOSUB 490: IF NOT FL THEN 1060
1055 PRINT "THIS IS AN ILLEGAL RECORD !!!": FOR J=1 TO
     500: NEXT J: PRINT CHR$(7,7): GOTO 980
1060 RETURN
1070 POP : GOTO 300
```

Fig. 8-13—*cont.* Program to update a sequential text file in Integer BASIC.

converted. Line 115 in any of the three figures is the most basic of the changes.

In this line we will dimension the string memory necessary for the FILENAME, NAME, SOCIAL SECURITY NUMBER, and DEPARTMENT. Since the PAY RATE is to be a "real number," and Integer BASIC doesn't allow real numbers, we will have to hold the real number value in a string variable, and use checks to ensure that the user is inputting a proper value. Line 115 in Fig. 8-13 shows how many characters can be input into the various string variables. This is only a minor limitation since Applesoft is limited to 255 characters per string anyway.

Two primary examples of changing from Applesoft to Integer BASIC are seen in lines 780–788 in Fig. 8-13 and 390–398 in Fig. 8-14. Applesoft has an ASSIGNED GOTO and Integer BASIC does not have an ASSIGNED GOTO statement, so this must be replaced by single IF-THEN statements. In the UPDATE program, when a record is inserted, the current record is concatenated and stored in a temporary variable. This is done in both programs but Applesoft and Integer BASIC string handling techniques are much different. Applesoft has built-in functions while Integer BASIC is at a much simpler level, but the same thing can be accomplished in both languages. For further study, see Fig. 8-13, lines 790–800 to see concatenation and lines 890–950 to see the original string reconstructed. The search of a string to see if it contains a selected token is similarly shown (three times) in Fig. 8-14 in lines 540–670.

If the READ program is to be able to select the records in the file by pay range, then the string value in RP$ must be converted to some kind of

```
10 DIM CHR$(27)
20 FOR J=1 TO 27: POKE 2054+J,J+128: NEXT J
30 POKE 2081,30
40 GOTO 100
50 CI= PEEK (37):CH= PEEK (36)
51 POKE 37,CI: POKE 36,CH: CALL -958: INPUT " ",NUM$

52 L= LEN(NUM$): IF L=0 THEN GOTO 51
53 D=0
54 FOR I=1 TO L
55 IF NUM$(I,I)=" " THEN GOTO 51
56 IF NUM$(I,I)#"." THEN GOTO 58
57 IF D#0 THEN GOTO 51:D=I: GOTO 59
58 IF ASC(NUM$(I,I))<176 OR ASC(NUM$(I,I))>185 THEN
   GOTO 51
59 NEXT I: IF (D=0 AND L>8) OR (D#0 AND L-D>2) THEN
   51: GOTO 61
60 L= LEN(NUM$):D=L-2
61 CNTS=0:BDOL=0:SDOL=0:DIS$="............."
62 J=D-1: IF D=0 THEN J=L
63 K=12-J-(J-1)/3:DIS$(K-1)="$": IF D=1 THEN 70
64 FOR I=1 TO J
65 IF K MOD 4>0 THEN 67
66 DIS$(K)=",":K=K+1
67 DIS$(K)=NUM$(I,I): IF (J-I)/4 THEN 68:SDOL=SDOL*
   10+ ASC(NUM$(I,I))-176: GOTO 69
68 BDOL=BDOL*10+ ASC(NUM$(I,I))-176
69 K=K+1: NEXT I
70 DIS$(12)="."
71 IF D>0 AND D<L THEN 72:DIS$(13)="00": GOTO 74
72 IF L-D<2 THEN 73:DIS$(13)=NUM$(D+1):CNTS=( ASC(NUM$
   (D+1,D+1))-176)*10+ ASC(NUM$(D+2,D+2))-176: GOTO
   74
73 DIS$(13)=NUM$(D+1):DIS$(14)="0":CNTS=( ASC(NUM$(
   D+1,D+1))-176)*10
74 RETURN
75 IF VAL#1 THEN 76:AP$="1": RETURN
76 IF VAL#2 THEN 77:AP$="2": RETURN
77 IF VAL#3 THEN 78:AP$="3": RETURN
78 IF VAL#4 THEN 79:AP$="4": RETURN
79 IF VAL#5 THEN 80:AP$="5": RETURN
80 IF VAL#6 THEN 81:AP$="6": RETURN
81 IF VAL#7 THEN 82:AP$="7": RETURN
82 IF VAL#8 THEN 83:AP$="8": RETURN
83 IF VAL#9 THEN 84:AP$="9": RETURN
84 AP$="0": RETURN
85 NUM$="":TVAL=BDOL:J=0: IF BDOL=0 THEN 89
86 FOR I=1 TO 4
87 VAL=TVAL/10 ^ (4-I): GOSUB 75
88 NUM$(I)=AP$:TVAL=TVAL-VAL*10 ^ (4-I): NEXT I:J=4

89 TVAL=SDOL: IF SDOL=0 AND J=0 THEN 94: IF SDOL>0 THEN
   90:NUM$(J+1)="0000": GOTO 93
90 FOR I=1 TO 4
91 VAL=TVAL/10 ^ (4-I): GOSUB 75
92 NUM$(I+J)=AP$:TVAL=TVAL-VAL*10 ^ (4-I): NEXT I
```

Fig. 8-14. Program to read a sequential text file in Integer BASIC.

```
 93 J=J+4
 94 NUM$(J+1)=".":TVAL=CNTS: IF CNTS>0 THEN 95:NUM$(
    J+2)="00": GOTO 97
 95 FOR I=1 TO 2:VAL=TVAL/10 ^ (2-I): GOSUB 75
 96 NUM$(I+J+1)=AP$:TVAL=TVAL-VAL*10 ^ (2-I): NEXT I
 97 J= LEN(NUM$): FOR I=1 TO J: IF NUM$(I,I)#"0" AND
    NUM$(I,I)#"." THEN 98: NEXT I: RETURN
 98 IF I>1 THEN NUM$=NUM$(I,J): RETURN
100 REM :PROGRAM TO READ A              SEQUENTIAL FILE
110 D$=CHR$(4,4)
115 DIM NUM$(11),DIS$(14),F$(30),NA$(200),SS$(12),DP$
    (20),RP$(20),L$(255),QU$(255),TN$(200),TS$(12),DT$
    (20),SDIS$(14)
120 SL=6: REM :SLOT # IS 6
130 DR=1: REM :DRIVE # IS 1
140 CALL -936: VTAB 3
150 PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
160 PRINT
170 INPUT "CHANGE THESE VALUES(Y OR N)? ",QU$
180 PRINT
190 IF QU$#"Y" THEN 240
200 INPUT "ENTER SLOT (1-7) ",SL
210 IF SL<1 OR SL>7 THEN 200
220 INPUT "ENTER DRIVE (1-2) ",DR
230 IF DR<1 OR DR>2 THEN 220
240 PRINT : INPUT "ENTER FILE NAME? ",F$
250 IF LEN(F$)>0 THEN 260
255 PRINT : PRINT "*** NAME IS INVALID ***": GOTO 240

260 FOR J=1 TO LEN(F$): IF ASC(F$(J,J))< ASC(CHR$(1,
    1)) OR ASC(F$(J,J))> ASC(CHR$(26,26)) THEN 270
265 PRINT "PLEASE DON'T TRY CONTROL CHARACTERS ": FOR
    J=1 TO 500: NEXT J: GOTO 240
270 NEXT J: PRINT "THE NAME IS GOOD!!!"
280 PRINT D$;"OPEN ";F$;",S";SL;",D";DR
290 CALL -936: PRINT "ENTER VALUE FOR SELECTION ELSE"

300 PRINT : PRINT "PRESS 'RETURN' TO LIST ALL ENTRIES"

310 PRINT : INPUT "ENTER NAME TOKEN ?",TN$
320 PRINT : INPUT "ENTER SS NUM TOKEN? ",TS$
330 PRINT : INPUT "ENTER DEPT TOKEN? ",DT$
340 CALL -936: PRINT : PRINT "ENTER RANGE OF PAY RATE<R>
    "
350 PRINT : PRINT : PRINT "SELECT ABOVE MINIMUM<M> "

360 PRINT : PRINT : PRINT "SELECT BELOW MAXIMUM<L> "

370 PRINT : PRINT : PRINT "ACCEPT ALL RATES<A> "
380 PRINT : PRINT : INPUT "ENTER SELECTION ?",QU$
390 IF QU$="M" THEN 400
392 IF QU$="L" THEN 430
394 IF QU$="R" THEN 460
396 IF QU$="A" THEN 490
398 GOTO 340
```

Fig. 8-14—*cont.* Program to read a sequential text file in Integer BASIC.

```
400 CALL -936: PRINT : PRINT "ENTER MINIMUM RATE ";:
    GOSUB 50:MBDOL=BDOL:MSDOL=SDOL:MCNTS=CNTS
410 UBDOL=10000:USDOL=0:UCNTS=0: GOTO 500
430 CALL -936: PRINT "ENTER UPPER LIMIT ";: GOSUB 50
    :UBDOL=BDOL:USDOL=SDOL:UCNTS=CNTS
440 MBDOL=0:MSDOL=0:MCNTS=0: GOTO 500
460 CALL -936: PRINT "ENTER BASE FIGURE ";: GOSUB 50
    :MBDOL=BDOL:MSDOL=SDOL:MCNTS=CNTS
470 PRINT : PRINT "ENTER MAXIMUM FIGURE ";: GOSUB 50
    :UBDOL=BDOL:USDOL=SDOL:UCNTS=CNTS
480 IF MBDOL>UBDOL THEN 460: IF MBDOL<UBDOL THEN 500

483 IF MSDOL>USDOL THEN 460: IF MSDOL<USDOL THEN 500

486 IF MCNTS>UCNTS THEN 460
488 GOTO 500
490 UBDOL=10000:USDOL=0:UCNTS=0:MBDOL=0:MSDOL=MBDOL:
    MCNTS=MSDOL
500 CALL -936
510 PRINT D$;"READ ";F$
520 INPUT NA$,SS$,DP$,RP$:NUM$=RP$: GOSUB 60:SDIS$=DIS$

530 IF NA$="END RECORD" AND SS$="000-00-0000" AND DP$
    ="00" AND RP$="00.00" THEN 760
535 PRINT D$;"IN#0"
540 L= LEN(TN$): IF L=0 THEN 590
550 IF LEN(NA$)<L THEN 750
560 FOR J=1 TO LEN(NA$)-L+1
570 IF TN$=NA$(J,J+L-1) THEN 590
580 NEXT J: GOTO 750
590 L= LEN(TS$): IF L=0 THEN 640
600 IF LEN(SS$)<L THEN 750
610 FOR J=1 TO LEN(SS$)-L+1
620 IF TS$=SS$(J,J+L-1) THEN 640
630 NEXT J: GOTO 750
640 L= LEN(DT$): IF L=0 THEN 690
650 IF LEN(DP$)<L THEN 750
660 FOR J=1 TO LEN(DP$)-L+1
670 IF DT$=DP$(J,J+L-1) THEN 690
680 NEXT J: GOTO 750
690 IF BDOL>UBDOL THEN 750: IF BDOL<UBDOL THEN 695
691 IF SDOL>USDOL THEN 750: IF SDOL<USDOL THEN 695
692 IF CNTS>UCNTS THEN 750
695 IF BDOL<MBDOL THEN 750: IF BDOL>MBDOL THEN 700
696 IF SDOL<MSDOL THEN 750: IF SDOL>USDOL THEN 700
697 IF CNTS<MCNTS THEN 750
700 PRINT "NAME    = ";NA$
710 PRINT "S SEC #= ";SS$
720 PRINT "DEPT    = ";DP$
730 PRINT "RATE    = ";SDIS$
740 PRINT : PRINT "---------": PRINT
750 GOTO 510
760 PRINT D$;"IN#0"
770 END
```

Fig. 8-14—*cont.* Program to read a sequential text file in Integer BASIC.

numeric form, or standardized so that a string comparison will make some sense. Comparing "123.45" to "1234.56", for example, will not make sense as far as a less than/equal to/ greater than test goes. The solution chosen for this program is to take the string and break it into three values and store them in three variables. For example, the number "12345678.90" will be broken in the following manner: value "1234" will be placed into a variable called BDOL, the value "5678" will be placed into a variable called SDOL, and the cents value of "90" is stored into CNTS. This means any value less than 100 million dollars can be stored and manipulated in three variable locations. The subroutine to do this is at lines 50–74 in Figs. 8-13 and 8-14. This will accept a string of 11 characters and check to ensure that it is "legal." This means no alphabetic characters, at most one decimal point, and the whole number must be less than nine characters in length, while the decimal part, if any, must be two characters or less. If the string (NUM$) passes inspection then the conversion to numeric variable starts (line 61). This not only converts to numeric form, but also creates a display form (DIS$) that is nice and neat for output on a financial statement. If the string to be converted is already in good form (has a decimal point and two decimal places, no alphabetics, proper length), then putting the string into NUM$ and GOSUB 60 will save time by bypassing the checks. The input lets the user put the type of input label (for example, "RATE = ", "AMOUNT = ?", etc.) before the GOSUB 50 is executed and if the actual string input is rejected, then the cursor will be returned to just after that input label. Lines 61 through 74 do the actual conversion to numeric and string form.

The subroutine at lines 85–98 will take the values BDOL, SDOL and CNTS and convert them into a standard form in NUM$ for writing to DUMMY file. This is essential for the READ program to be able to make sense out of the PAY RATE string input from the data file. This uses a supporting subroutine at lines 75–84 which determines if the conversion value is 1–9 and returns the string equivalent. If the value is not one of these, then "0" is automatically returned.

In the UPDATE program, RP$ is read from the file and converted in line 360. When a record is added or inserted, or added to the end of the file, it is input through line 51 and checked (lines 660 and 850). If an item is altered, the option of no entry (meaning no change) must be allowed so the lines 1040–1045 input the initial value, and then start in at line 52. If the user enters a value that is not a legal entry, the subroutine will not continue until a good value has been entered.

In the UPDATE program (Fig. 8-13), lines 660 and 850 illustrate the

process of preparing the PAY RATE value for output to the DUMMY file. The three values of BDOL, SDOL, and CNTS were not changed between the inputting of RP$ and the outputting of RP$.

In the READ program in Fig. 8-14, the selecting of the PAY RATEs for the search is shown in lines 400–490. The upper limit is kept in three variables: UBDOL, USDOL, and UCNTS, and the lower limit is kept in

Table 8-1. Variable Dictionary for Create, Update, and Read Programs

Command	Definition
A	Accepts all rates (option).
ASC	Converts a string character into an ASCII value. ASC("A") = 65
CHR$(7)	Returns the ASCII that corresponds to the value of the argument. Rings the bell on the computer.
CP	Vertical cursor position.
D$	D$ = CHR$(4). This statement assigns Control D to D$.
DP$	String to hold the department code.
DR	Disk drive variable.
DT$	Holds the department code token.
F$	String to hold the file name.
FL	Flag value variable.
J	Loop variable.
L	Variable to hold the length of a string.
L$	Temporary variable into which the record is placed.
L0	Temporary variable that holds the length of the record.
L1	Temporary variable that holds the length of the name field string.
L2	Temporary variable that holds the length of the social security field string.
L3	Temporary variable that holds the length of the department code field string.
LEN	Basic word that returns the number of characters in a string.
LR	Low rate of pay rate.
M	Above a certain minimum (option).
NA$	Holds the name string.
POP	Causes a jump out of a subroutine without going through the RETURN statement.
PRINT D$;"IN#0"	Returns input control to the keyboard.
PRINT D$;"PR#0"	Returns output control to the CRT.
QU$	Question string into which the strings are placed.
R	Range of pay rates (option).
RP	Rate of pay.
STR$(RP)	Converts the numeric value held in the rate of pay variable into a string value.
TN$	Variable to hold the name string token.
TS$	Variable to hold the social security number token.
UR	Upper range of the pay rate.
VAL(QU$)	VAL converts a string into a numeric value.

MBDOL, MSDOL, and MCNTS. If the user is to enter one of these values (for example, the upper limit as in line 430), the label "ENTER UPPER LIMIT" is printed, then a GOSUB 50 is executed and then UBDOL is set to BDOL and USDOL to SDOL and UCNTS to CNTS.

To determine whether the record just read from memory is acceptable by the value of the pay rate, lines 690–697 compare the upper and lower limit variables to the values just generated from lines 60–74 (the PAY RATE from the current record). Line 690 checks to see if the pay rate is greater than the upper limit. If it is, then the record is rejected. If it is equal or less, then the program defaults through to the lower limit check. The lower limit check is similar in lines 695–697. A variable dictionary for Create, Update, and Read programs is shown in Table 8-1.

chapter 9

Extending and Refining Sequential Files

In the two UPDATE programs in Chapter 8 the POP command was used to exit the CHANGE file subroutine abnormally and bypass writing the current record to the output file. In place of such a procedure, it is recommended that a flag be used. Such a procedure can be implemented by replacing or inserting the following three lines in either the Applesoft or Integer BASIC programs given in Chapter 8.

```
375 PF=1 : REM - PF=PRINT FLAG
390 IF PF THEN GOSUB 560
1070 PF=0 : RETURN
```

These changes will cause the UPDATE program to write the current record unless the PF value is set to zero in the DELETE section of the CHANGE subroutine.

In Applesoft, when writing a record to disk, there exists a better way to indicate that several items constitute a single record than to just have the variables in individual print statements. Such a method employs the Applesoft technique of breaking an input of characters at commas or colons, so that the various values may be put into different variables. The statement to do this is listed as follows:

```
570 PRINT NA$;",";SS$;",";DP$;",";RP
```

This will actually write a comma to disk. Since the comma replaces the return character, no more space is used on the disk.

In the CREATE program, the pay rate for the END OF FILE RECORD was created as "00.00" while in the UPDATE program it was read in into RP. This may seem to be a contradiction, but it is exactly the same when inputting values from the keyboard. The values coming from disk (or keyboard) are checked to make sure that only numeric values are being read into a numeric variable.

This means that any character coming from disk can go to a string variable, while the characters going into a numeric variable must be numeric!

USING THE UPDATE PROGRAM PROFICIENTLY

When the UPDATE program asks for a change in the file, you must understand the precedence that it uses to process the options at the user's command. The ACCEPT, CHANGE, and DELETE are all on the same level. One record is changed and then the subroutine is exited. The INSERT option is on a level higher than the other three commands.

This command will allow one or more insertions and then return to the record presented. Therefore, insertions must always be done before any of the other three functions to ensure the correct sequencing of records on the file.

When using the update program, it is up to the user to enter the records in the correct order. Ordering the records by alphabetical index on the employee name is possible, as is ordering by social security number, but this is up to the user to keep straight by the proper order of entry.

In the programs in Chapter 8, the use of control characters was disallowed. Tables 9-1 and 9-2 may give an indication of the reason for this. Table 9-1 is the Applesoft listing for control characters, Table 9-2 is the Integer BASIC listing. CTRL - M is exactly identical to pressing the RETURN key. The RETURN key is merely a convenience for the user. Trying to insert a Control-M in the name would merely result in the name being all the characters (if any) before CTRL - M is pressed. The forward arrows and back arrows are also the same as CTRL - U and CTRL - H respectively, and they can cause different results in the two different languages.

Restrictions of These Programs

As was mentioned in Chapter 8, the method of updating the file from the current to a dummy, then deleting the current and renaming the dummy will only allow half the disk space to be used for the file. To overcome this

Table 9-1. How Control Characters Affect the Disk Operating System (DOS) When Placed in a File Name* *Applesoft*

Control	Control Character Placed in Position #1	Control Character Placed in Positions #2 to #30
A	SYNTAX ERROR BREAK IN LINE XXX OF PROGRAM	DOES NOT AFFECT DOS
B	SYNTAX ERROR	DOES NOT AFFECT DOS
C	BREAK IN LINE XXX OF PROGRAM	DOES NOT AFFECT DOS
D	SYNTAX ERROR	DOES NOT AFFECT DOS
E	SYNTAX ERROR	DOES NOT AFFECT DOS
F	SYNTAX ERROR	DOES NOT AFFECT DOS
G	SYNTAX ERROR	DOES NOT AFFECT DOS
H	BACK ARROW	BACK ARROW
I	SYNTAX ERROR	DOES NOT AFFECT DOS
J	SYNTAX ERROR	DOES NOT AFFECT DOS AFFECTS HOW DISK CATALOG IS READ
K	SYNTAX ERROR	DOES NOT AFFECT DOS
L	SYNTAX ERROR	DOES NOT AFFECT DOS
M	SAME AS *RETURN*	SAME AS *RETURN*
N	SYNTAX ERROR	DOES NOT AFFECT DOS
O	SYNTAX ERROR	DOES NOT AFFECT DOS
P	SYNTAX ERROR	DOES NOT AFFECT DOS
Q	SYNTAX ERROR	DOES NOT AFFECT DOS
R	SYNTAX ERROR	DOES NOT AFFECT DOS
S	SYNTAX ERROR	DOES NOT AFFECT DOS
T	SYNTAX ERROR	DOES NOT AFFECT DOS
U	TREATED AS A SPACE IGNORED AT BEGINNING	TREATED AS A SPACE DOES NOT AFFECT DOS
V	SYNTAX ERROR	DOES NOT AFFECT DOS
W	SYNTAX ERROR	DOES NOT AFFECT DOS
X	DELETES A LINE	DELETES A LINE
Y	SYNTAX ERROR	DOES NOT AFFECT DOS
Z	SYNTAX ERROR	DOES NOT AFFECT DOS

*This table shows the result of using a CONTROL CHARACTER, either accidentally or on purpose, in a DISK FILE NAME. A CONTROL CHARACTER is produced by pressing CTRL and then an alpha character. A CONTROL CHARACTER is not printed on the screen, but is placed on disk. CONTROL CHARACTERS affect the DISK OPERATING SYSTEM (DOS).

handicap, the program must be modified to access multiple disk drives. This will enable the program to read the file from one disk drive and write to the dummy on the second unit. This will allow the file that is being read in order to update to be saved for future reference, or in case the updated file and all its many copies happen to be destroyed. Modifications to create such a program will be given in this chapter under Multidrive Operation.

Another minor limitation is the concatenation of the record variables that

Table 9-2. How Control Characters Affect the Disk Operating System (DOS) When Placed in a File Name* *Integer Basic*

Control	Control Character Placed in Position #1	Control Character Placed in Positions #2 to #30
A	SYNTAX ERROR BREAK IN LINE XXX OF PROGRAM	DOES NOT AFFECT DOS
B	SYNTAX ERROR	DOES NOT AFFECT DOS
C	STOPS PROGRAM	STOPS PROGRAM
D	SYNTAX ERROR	DOES NOT AFFECT DOS
E	SYNTAX ERROR	DOES NOT AFFECT DOS
F	SYNTAX ERROR	DOES NOT AFFECT DOS
G	SYNTAX ERROR	DOES NOT AFFECT DOS
H	BACK ARROW	BACK ARROW
I	SYNTAX ERROR	DOES NOT AFFECT DOS
J	SYNTAX ERROR	DOES NOT AFFECT DOS
K	SYNTAX ERROR	DOES NOT AFFECT DOS
L	SYNTAX ERROR	DOES NOT AFFECT DOS
M	SAME AS *RETURN*	SAME AS *RETURN*
N	SYNTAX ERROR	DOES NOT AFFECT DOS
O	SYNTAX ERROR	DOES NOT AFFECT DOS
P	SYNTAX ERROR	DOES NOT AFFECT DOS
Q	SYNTAX ERROR	DOES NOT AFFECT DOS
R	SYNTAX ERROR	DOES NOT AFFECT DOS
S	SYNTAX ERROR	DOES NOT AFFECT DOS
T	SYNTAX ERROR	DOES NOT AFFECT DOS
U	FORWARD ARROW IGNORED AT THE BEGINNING	GIVES A SPACE IN THE FILE NAME DOES NOT AFFECT DOS
V	SYNTAX ERROR	DOES NOT AFFECT DOS
W	SYNTAX ERROR	DOES NOT AFFECT DOS
X	DELETES A LINE	DELETES A LINE
Y	SYNTAX ERROR	DOES NOT AFFECT DOS
Z	SYNTAX ERROR	DOES NOT AFFECT DOS

*This table shows the result of using CONTROL CHARACTERS, either accidentally or on purpose, in a DISK FILE NAME. A CONTROL CHARACTER is produced by pressing CTRL and then an alpha character. A CONTROL CHARACTER is not printed on the screen, but is placed on disk. CONTROL CHARACTERS affect the DISK OPERATING SYSTEM (DOS).

is done to save the values in these variables while new records are being inserted. Such a procedure will limit the total length of the record to 255 characters—this was done in order to use the least amount of memory possible. Using a single temporary variable for each record variable will extend the possible length, and should create much simpler program segments before and after the insertion of new records.

Finally, the single drive version of this program can be very dangerous to use if the employee file gets close to the limit of half the disk space. The

only way of finding out whether there is not enough room left will be an error message that causes the program to stop. The entire run must be redone with a multiple drive configuration. If this comes as a complete surprise to the user, then it is bound to be very disturbing.

Flexibilities of These Programs

In the UPDATE program, when social security number or department is entered, there are no limitations whatsoever placed upon the values entered. For example, a legitimate entry for social security number should be "XXX-XX-XXXX", where the X's represent any number from 0 to 9. Since there are no limitations on the values entered, then the social security field can be used for any string value for which the user has a need. The same thing can be said of the department string entry.

```
315 LN =  LEN (TN$)
325 LS =  LEN (TS$)
335 LT =  LEN (DT$)
535 L1 =  LEN (NA$):L2 =  LEN (SS$):L3 =  LEN (DP$)
540  IF LN = 0 THEN 590
550  IF L1 < LN THEN 750
560 L = L1 - LN + 1: FOR J = 1 TO L
570  IF TN$ =  MID$ (NA$,J,LN) THEN 590
590  IF LS = 0 THEN 640
600  IF L2 < LS THEN 750
610 L = L2 - LS + 1: FOR J = 1 TO L
620  IF TS$ =  MID$ (SS$,J,LS) THEN 640
640  IF LT = 0 THEN 690
650  IF L3 < LT THEN 750
660 L = L3 - LT + 1: FOR J = 1 TO L
670  IF DT$ =  MID$ (DP$,J,LT) THEN 690
```

(A) Applesoft.

```
315 LN= LEN(TN$)
325 LS= LEN(TS$)
335 LT= LEN(DT$)
535 L1= LEN(NA$):L2= LEN(SS$):L3= LEN(DP$)
540 IF LN=0 THEN 590
550 IF L1<LN THEN 750
560 L=L1-LN+1: FOR J=1 TO L
570 IF TN$=NA$(J,J+LN-1) THEN 590
590 IF LS=0 THEN 640
600 IF L2<LS THEN 750
610 L=L2-LS+1: FOR J=1 TO L
620 IF TS$=SS$(J,J+LS-1) THEN 640
640 IF LT=0 THEN 690
650 L=L3-LT+1: FOR J=1 TO L
660 IF DT$=DP$(J,J+LT-1) THEN 690
```

(B) Integer BASIC.

Fig. 9-1-Program to be incorporated directly in the program in Fig. 8-9.

The only action necessary to revise these programs to accommodate different kinds of data, such as name, address, or customer identification, etc., would be to change all labels displayed for input purposes, and possibly to change the special END OF FILE RECORD.

SPEED IMPROVEMENTS

The main speed improvement suggestion is for the READ program. Each time a record is checked, the token length for the appropriate record is placed in a single temporary variable. If each token length were placed in a unique variable then this repetitious process would be eliminated. Also, the length of the string record being searched is used in the loop in the form LEN( ), and evaluating this each time the loop is executed is very time consuming. Placing these string lengths into temporary variables when the field is going to be searched would also reduce the time used to evaluate a record.

The program was written in order to use the least amount of memory possible. This is the reason for the steps taken in the programs given in Chapter 8. Alterations to include these features are given in Fig. 9-1. To incorporate these changes into the READ program, merely load the program and type the lines listed in Fig. 9-1.

Lines 315–335 store the length of the token value entries, while 535 stores the lengths of the variables in the record being processed.

The lines 540–670 are replacement lines that utilize the values stored in the previous lines.

MULTIDRIVE OPERATION

Multidrive operation requires very few changes in terms of software. The major change is the acquisition of the second unit if the user has only one disk drive. These changes are listed in Fig. 9-2. The only real change is to enter a destination slot and disk drive. This will allow the old file to be read in one drive, and to build a new file in the second drive. Such a program will not be usable in single drive operations because of the way in which the DOS handles the rename feature. As with Fig. 9-1, merely load the program and type in the lines in the figure and the modifications are complete.

This new capability needs to be carefully understood because once an update is through, the old file in the source drive is now the backup. The file in the second unit is the updated file, and will be read the next time the

```
125 SD = 6:DD = 2
163  PRINT "UPDATED FILE IN SLOT ";SD;" DRIVE ";DD
165  PRINT
200  INPUT "ENTER SOURCE SLOT (1-7) ?";SL
220  INPUT "ENTER SOURCE DRIVE (1-2) ?";DR
231  INPUT "ENTER DESTINATION SLOT (1-7) ?";SD
232  IF SD < 1 OR SD > 7 THEN 231
234  INPUT "ENTER DESTINATION DRIVE (1-2) ?";DD
235  IF DD < 1 OR DD > 2 THEN 234
237  IF SL = SD AND DR = DD THEN 120
272  PRINT : PRINT "DELETE OLD ";F$;" OFF ": INPUT "DE
     STINATION DRIVE (Y/N)? ";QU$: IF QU$ = "N" THEN 2
     80
274  HOME : PRINT "OPEN DOOR ON SOURCE DRIVE TO ": PRINT
     : PRINT "BE SURE THE GOOD FILE ISN'T DELETED": PRINT

276  INPUT "PRESS 'RETURN' WHEN READY !";QU$
278  PRINT D$;"UNLOCK ";F$;",S";SD;",D";DD: PRINT D$;"
     DELETE ";F$;",S";SD;",D";DD
279  PRINT "TASK COMPLETE ": PRINT : INPUT "CLOSE DRIV
     E DOOR !";QU$
290  PRINT D$;"OPEN DUMMY,S";SD;",D";DD
430  REM :NOT NEEDED!
440  PRINT D$;"CLOSE DUMMY"
450  PRINT D$;"RENAME DUMMY,";F$
460  PRINT D$;"LOCK ";F$;",S";SD;",D";DD
470  PRINT D$;"CLOSE "
475  PRINT "UPDATED FILE IS IN SLOT ";SD;" DRIVE ";DD
```

(A) Fits in Fig. 8-5 (Applesoft).

```
200 INPUT "ENTER SOURCE SLOT (1-7) ? ",SL
220 INPUT "ENTER SOURCE DRIVE (1-2) ?",DR
231 INPUT "ENTER DESTINATION SLOT (1-7) ?",SD
232 IF SD<1 OR SD>7 THEN 231
234 INPUT "ENTER DESTINATION DRIVE (1-2) ?",DD
235 IF DD<1 OR DD>2 THEN 234
237 IF SL=SD AND DR=DD THEN 120
290 PRINT D$;"OPEN DUMMY,S";SD;",D";DD
430 REM :NOT NEEDED!
440 PRINT D$;"CLOSE DUMMY"
450 PRINT D$;"RENAME DUMMY,";F$
460 PRINT D$;"LOCK DUMMY,S";SD;",D";DD
470 PRINT D$;"CLOSE"
475 PRINT "UPDATED FILE IS IN SLOT ";SD;" DRIVE ";DD
```

(B) Fits in Fig. 8-14 (Integer BASIC).

Fig. 9-2. Program to convert a single disk drive system to a multidrive disk.

file needs to be updated. This means that, if through careless procedure, the old file is used instead of the new file, then all the changes made will be lost, and the user will wonder what happened to the file. Also, for a disk to be used as output after it had held the input file, the locked file needs to be

taken off before the new file is to be put on it. This is up to the user to ensure that the destination is cleaned of any extraneous files.

Line 125 will create variables for the destination slot and drive. Lines 163–165 present the default values for the destination file, Fig. 9-2.

Lines 200–235, Fig. 9-2, are new displays when the default values are changed. This will make the differences in the two sets of questions more understandable.

Lines 272–279, Fig. 9-2, present the option to remove an old file of the same name off the destination disk. This will save time when an old disk is reused (as long as the old file need not be kept for the company's records or tax or governmental documentation). The procedure even halts program execution to let the user open the door on the source drive, to ensure that the good file is not deleted in case a mistake is made in placing the diskettes in the drives.

Line 290, Fig. 9-2, is the new open command to put the DUMMY file on the destination drive.

The new lines 430–470, Fig. 9-2, reflect the change in actions necessary to successfully terminate the program. Line 475, Fig. 9-2, displays a reminder to the user to make sure that the updated file is removed from the drive and put in a correct place for the current employee file.

MULTISECTION FILES

Sooner or later, the user will have a file that fills a full disk and still doesn't have enough room to fit. Then the file must be sectioned, that is, split into multiple sections until the file is complete.

This condition requires that a section header and an end of section marker, as well as an end of file record, be implemented. This means, that if section two is placed in the source drive when section one is needed, an informative message is displayed and requires the user to replace section two with section one.

When the end of section is encountered, the program will ask that the next section be placed in the source slot and when this is acknowledged the program continues. A similar action will take place when the destination disk is filled to capacity. The program will write an end of section marker, close the file, and ask for another disk in the destination drive for the next section. To determine when the destination slot is full, a simple counting of the number of bytes written to the destination disk can be done. Using the LEN function, the exact number of bytes per item in each record can be counted.

The section header could include some kind of date record that would be used by the program, to verify whether or not the file from the last update is being used as the source file. This can eliminate many problems concerning incorrect updates.

To the purist, that is, the person who thinks one program should work under every situation, the previous paragraphs must seem shocking. Now you have come into the real world of programming. The programmer must be given some indication as to the size of the file to be implemented. A file with 50 records could all be held in memory. A file with 50,000 records could not be handled in the same manner (not on this machine at any rate). If these modifications seem suited to this machine (or any other minicomputer), then why do very large data processing sites have to do exactly similar kinds of procedures? The answer is that these are only finite machines and the only thing that is infinite about them is the programmer's skill in overcoming such limitations.

chapter 10

Multisection Sequential File Date Updates

VOCABULARY

In business data processing, the distinction between the old file and the current file can be confusing to the uninitiated. In this book, the file updates are discussed in the following manner. The file name is TEST. The old file TEST is updated to become the new file TEST. The file update begins.

1. Start with the old file—CURRENT MASTER "TEST".
2. CURRENT MASTER "TEST" is temporarily called the INPUT MASTER "TEST".
3. The new records are typed in and the update program is RUN.
4. The RUN creates the OUTPUT MASTER "TEST".
5. The OUTPUT MASTER "TEST" is renamed CURRENT MASTER "TEST".
6. The next update of the file begins at step No. 1, with the CURRENT MASTER "TEST", and repeats the process.

WORD—A word is a phrase or group of phrases that encompasses a specific meaning. A "WORD" is composed of (A) the beginning of section phrases, (B) end of file record, and (C) end of section phrases. The begin-

ning of section "WORD", end of file "WORD", and end of section "WORD", are illustrated in the following outline.

A. BEGINNING OF SECTION WORD.
   1. "SECTION INDICATOR"
   2. SECTION NUMBER
   3. "DATE RECORD"
   4. DATE OF UPDATE
B. END OF FILE WORD.
   1. END RECORD
   2. 000–000–0000 (for SOCIAL SECURITY #)
   3. 00 (for DEPARTMENT)
   4. 00.00 (for PAY RATE)
C. END OF SECTION WORD.
   1. END SECTION, 000–00–0000,00,0.

CURRENT MASTER FILE—The CURRENT MASTER FILE is the file that was created when the update program was last run. The updated file is referred to as the INPUT (UPDATED) MASTER FILE.

DATE INPUT—In the programs in Figs. 10-1 and 10-3, the date is input in six digit format. The form (MMDDYY) requires that the current date be input as 121581 for December 15, 1981.

DATE OF RECORD—The date of record is the date that the update was last run, and is stored on each section of the CURRENT MASTER FILE.

DATE OF ENTRY—The date of entry is today's date and is stored on the OUTPUT (UPDATED) MASTER FILE.

DESTINATION DRIVE—The destination drive is the disk drive where the new OUTPUT (UPDATE) MASTER file is built.

FILE SECTION—A file section is a group of records that fill the storage area on a disk. If there are more records in the file than can be placed on the disk, the section must be closed, and another section opened on another disk. Sectioning is a method to maintain file continuity on a limited storage medium.

INPUT MASTER—The INPUT MASTER file is the name temporarily given to the CURRENT MASTER file while the update program is RUNning.

SECTION HEADER—The section header is composed of four parts, (1) the "WORD" section indicator, (2) the section number, (3) the word "DATE RECORD", and (4) the current date.

SECTION NUMBER—The section number is a numeric digit that iden-

```
100  REM :PROGRAM TO CREATE A    SEQUENTIAL FILE
110 D$ =  CHR$ (4)
120 SL = 6: REM :SLOT IS #6
130 DR = 1: REM :DRIVE IS #1
140  HOME : VTAB 3
150  PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
160  PRINT
170  INPUT "CHANGE THESE VALUES(Y OR N)? ";QU$
180  PRINT
190  IF QU$ <  > "Y" THEN 240
200  INPUT "ENTER SLOT (1-7) ?";SL
210  IF SL < 1 OR SL > 7 THEN 200
220  INPUT "ENTER DRIVE (1-2) ?";DR
230  IF DR <  > 1 AND DR <  > 2 THEN 220
240  PRINT : INPUT " ENTER FILE NAME ?";F$
250  IF  LEN (F$) = 0 THEN  PRINT : PRINT "*** NAME IS
      INVALID ***": GOTO 240
260  FOR J = 1 TO  LEN (F$): IF  ASC ( MID$ (F$,J,1)) <
     32 THEN  PRINT : PRINT "*** PLEASE DON'T TRY CONT
     ROL CHARA-": PRINT "CTERS YET !": GOTO 240
270  NEXT J: PRINT : PRINT "!!! THE NAME IS GOOD"
280  PRINT D$;"OPEN ";F$;",S";SL;",D";DR
285  GOSUB 400
290  PRINT D$;"WRITE ";F$
295  GOSUB 450
300  PRINT "END RECORD": PRINT "000-00-0000": PRINT "0
     0": PRINT "00.00"
310  PRINT D$;"CLOSE"
320  PRINT D$;"LOCK ";F$;",S";SL;",D";DR
330  END
400  HOME : INPUT "ENTER CREATION DATE (MODAYR) ?";QU$

410  IF  LEN (QU$) <  > 6 THEN 400
420  GOSUB 500
430  IF DF THEN 400
440  RETURN
450  PRINT "SECTION INDICATOR": PRINT "1"
460  PRINT "DATE RECORD": PRINT QU$
490  RETURN
500 DF = 0:L1 =  VAL ( LEFT$ (QU$,2)):L2 =  VAL ( MID$
     (QU$,3,2)):L3 =  VAL ( RIGHT$ (QU$,2))
510  IF L2 < 1 OR L2 > 31 THEN 560
520  IF L1 = 1 OR L1 = 3 OR L1 = 5 OR L1 = 7 OR L1 = 8
      OR L1 = 10 OR L1 = 12 THEN 570
530  IF L2 > 30 THEN 560
540  IF L1 <  > 2 THEN 570
550  IF L2 <  = 28 + ( INT (L3 / 4) * 4 = L3) - (L3 =
     0) THEN 570
560 DF = 1
570  RETURN
```

Fig. 10-1. Program to create a DUMMY input sequential file with a section header, section number, date record word, and the date the file was created.

tifies the file section. The section number begins with the digit one (1), and is incremented by one (1) for each additional section.

SOURCE DRIVE—The source drive is the drive into which the CURRENT MASTER file is placed.

UPDATE—Update is a method to modify the CURRENT MASTER file into an OUTPUT (UPDATED) MASTER file.

Previous chapters of this book have presented programs to CREATE, UPDATE, and READ SEQUENTIAL FILES, to increase the speed at which these programs run, and accommodate multiple disk drives. Chapter 10 presents the culmination of sequential files by presenting functional programs to CREATE, UPDATE, and READ SEQUENTIAL FILES, to use multiple disk drives, to create files with multiple sections, and to use the current date to update the files.

MULTISECTION FILES

All disk space is finite, whether it is 1000 megabytes, as in large disk storage spaces, or approximately 146 kilobytes, as in the Apple DOS 3.3 disk.

When files are stored on disks, the files must somehow fit the available disk storage space. If the files are larger than the disk storage space, the files must be ended on one disk and the file continued on another disk. To do this, the file must be broken into sections, or sectionalized.

The 5¼ inch disk used with DOS 3.3 will store approximately 146,000 bytes. (The Apple computer has eight bits to the byte.) If there are 1000 records to be stored, and each record contains 200 bytes, the 200,000 bytes will not fit on the 146,000 byte storage disk.

How a Section Begins

The section is opened on the storage disk when the program in Fig. 10-1 is RUN.

The subroutine at lines 450 through 490 writes the section header and section number, "SECTION INDICATOR" " 1". It also writes the "word", DATE RECORD, and the date of entry of the record, 121581 (QU$).

How the Section Is Closed

The section is closed in one of two ways, (1) when running the program in Fig. 10-1, the end of record word is placed at the end of the file to create

the dummy program, and (2) when the program in Fig. 10-3 is run to place either an end of file word or an end of section word when a disk is full.

The section is closed by running the program in Fig. 10-1, by lines 300 through 330. Since there are no records in the dummy file, the file is closed by the end of file word. The end of file word consists of (1) END RECORD, (2) 000-00-0000 FOR SOCIAL SECURITY NUMBER, (3) 00 FOR DEPARTMENT, and (4) 00.00 FOR PAY RATE.

The section is closed off by running the program in Fig. 10-3 in one of two ways. If the file does not fill the storage space on the disk, the section will be closed by the end of file word, in lines 300 through 330 (Fig. 10-3). If the file fills the storage space on the disk, the file will be closed by the end of section word, line 480. Line 480 calls two subroutines. The subroutine at lines 10210 through 10240 closes the file, and the subroutine at lines 10000 through 10200 allows the next file section to be put in the disk drive.

The section number and the date record must be in the file because the

```
]MON C,I,O

]RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)? N


 ENTER FILE NAME ?TEST

!!! THE NAME IS GOOD
OPEN TEST,S6,D1
ENTER CREATION DATE (MODAYR) ?121581
WRITE TEST
SECTION INDICATOR
1
DATE RECORD
121581
END RECORD
000-00-0000
00
00.00
CLOSE
LOCK TEST,S6,D1

]PR#0
```

Fig. 10-2. DUMMY record created when the program in Fig. 10-1 is RUN.

program in Fig. 10-3 checks to determine if the correct section has been placed in the source drive.

The section markers close off the disk, so the records are properly closed on one disk, and the next file record is opened on the next disk with the next section marker, and the same date header. The section markers also close the disk so the DISK FULL error message is avoided.

The program in Fig. 10-1 is used to create a dummy file with no data, and the section header, with its four component parts, (1) the section indicator, (2) the section number, (3) the section word "DATE RECORD", and (4) the current date. It is necessary to create the dummy file, so when the program in Fig. 10-3 is RUN, the section header is available. The program in Fig. 10-3 checks and verifies the date of record of the CURRENT MASTER file (dummy), in order to determine if the CURRENT MASTER file is in the source drive.

When the program to create a dummy sequential file, Fig. 10-1, is RUN interactively with the user, the dummy file is created, Fig. 10-2. If the disk on which the file is to be created is in the correct disk drive, the values of the slot and drive option are not changed. The program then asks the user to "ENTER FILE NAME ?" The program checks to determine if the file name entered was legal, lines 250 through 280. The DUMMY file was written to the disk in slot #6, drive #1.

Chart 10-1 is the variable dictionary for the CREATE (Fig. 10-1), UPDATE (Fig. 10-3), and READ (Fig. 10-6) programs.

The UPDATE program, Fig. 10-3, is a functional program that can be used by a small business (up to 100 employees) to maintain employee records. The UPDATE program is the skeleton for a company payroll. A payroll program would need other categories such as hours worked, overtime, bonuses, withholding taxes, social security taxes, other deductions, and the necessary mathematical computations.

The CREATE, UPDATE, and READ programs in this chapter are very similar to the same programs in Chapter 8. The variables are similar, the program sections are similar; however, the line numbers are different. Fig. 10-3 is different from the program in Fig. 8-5 in that it has a routine to create file sections, a routine to check the program sections, a date input and a date check routine, and routines to tell the user which input and output files are to be placed in the source and destination disk drives.

The UPDATE program, Fig. 10-3, begins at line 100 with the comment, PROGRAM TO UPDATE A SEQUENTIAL FILE. Line 110 initializes D$ = CHR$(4), so D$ can be typed for use with the disk operating system. Line 120 is GOSUB 20000.

Chart 10-1. Variable Dictionary for Create, Update, and Read Programs

Command	Definition
A	Accepts all rates (option).
ASC	Converts a string character into an ASCII value.
BC	Byte count—initialized to 40 bytes.
BL	Byte length—300 bytes per section of the file.
C	Change the record (option).
CD$	Current date.
CHR$(7)	Returns the ASCII that corresponds to the value of the argument. Rings the bell on the computer.
CP	Vertical cursor position.
D$	D$ = CHR$(4). This statement assigns Control D to D$.
DD	Destination drive
DF	Date flag.
DR	The disk drive number of the input file.
DT$	Holds the department code token.
ES	Entered section—the section of the output file the program reads to help the user to determine if the old section of the file is needed. It asks the question, "DO YOU WANT TO DELETE IT?" Allows the user to change the disk if deletion is undesirable, lines 10060–10080.
F$	String to hold the file name.
FL	Flag value variable.
L	Variable to hold the length of a string.
L1	Value of the month stored in QU$, which has had a date entered.
L2	The day of the month stored in QU$.
L3	The two digits of the year stored in QU$.
LD$	The last date that the update program has been run. The date of entry date is stored on the file being read.
LEN	Basic word that returns the number of characters in the string.
LR	Low rate of pay rate.
M	Pay above a certain minimum (option).
NA$	Holds the name string.
PRINT D$;"IN#0"	Returns input control to the keyboard.
PRINT D$;"PR#0"	Returns output control to the CRT.
PF	Print flag—when set to zero (0) the record is deleted by not putting it to the output file. PF = 1 places the record in the file.
QU$	Question string into which the strings are placed.
R	Range of pay rates (option).
RP	Rate of pay.
SD	Source drive.
SL	Slot input.
SN	Section number of the input file needed.
SO	Section number of the output file.
STR$(RP)	Converts the numeric value held in the rate of pay variable into a string value.
TN$	Variable to hold the name string token.
TS$	Variable to hold the social security number token.
UR	Upper range of pay.
VAL(QU$)	VAL converts a string into a numeric value.

Lines 20000 through 20230 are the same as program lines 120 through 270 in Fig. 8-5. The subroutine runs from lines 20000 through 20350.

The file section routine is from lines 20240 through 20350. This routine creates the section length, the section input number, the date of the update, and the date of the last update.

In line 20420 the variable BL is initialized to 300 (BL = 300), which signifies that there are to be 300 bytes in each section of the file. The 300 byte length is a very small number compared to the 146000 bytes on a 5¼ inch 3.3 DOS disk. The 300 byte length sector is used as a method to test the program without typing in a great number of employee records.

In an actual program, the byte length (BL) should be set several hundred bytes less than the total byte length of the disk. For example, BL = 140000 in DOS 3.3, and BL = 120000 for DOS 3.2. The length of the file should be set several hundred bytes less than the disk storage capacity because an end of section word needs to be written before the file is ended.

If the section length exceeds the storage space on the disk, the DISK FULL error message stops the program.

If the disk storage space on the disk is exceeded and the program fails, it is necessary to recover at the beginning of the UPDATE program. To recover, press RESET, and RUN the UPDATE program. The INPUT MASTER file must be renamed using the immediate execution mode to place the file name TEST on the proper disk.

The immediate command, CATALOG Sx, Dw, and press RETURN is used to make certain the DOS reaches the disk that lists the INPUT MASTER file. If the INPUT MASTER file is in slot #6, drive #1, the command is CATALOG S6,D1. This command is preferred, so you are certain DOS is looking at the disk drive that contains the disk with the INPUT MASTER file. The immediate command RENAME INPUT MASTER, TEST, is then used.

```
CATALOG S6, D1
RENAME INPUT MASTER, TEST
```

In line 20270, Fig. 10-3, the program asks the user to "ENTER TODAY'S DATE (MMDDYY) ?";QU$. If the date input contains exactly six characters, the program defaults through line 20280 to line 20290, which is GOSUB 20360.

The subroutine at lines 20360 through 20430 sets the date flag to zero (DF = 0). The (MMDDYY) is input as a string. The string is broken up into L1, L2, and L3 in order to check the month, day, and year of the date. The date is checked against the calendar according to the old saying: Thirty

```
100  REM :PROGRAM TO UPDATE A    SEQUENTIAL FILE
110 D$ =  CHR$ (4)
120  GOSUB 20000
130  HOME : INPUT "PRESS 'RETURN' WHEN DRIVES ARE READ
     Y!";QU$
140  GOSUB 12010
150  GOSUB 10010
160  GOSUB 390
170  IF FL = 1 THEN 300
180  IF FL = 0 THEN 200
190  GOSUB 12180: GOSUB 12000: GOTO 160
200  HOME
210  PRINT "NAME =";  SPC( 4);NA$
220  PRINT : PRINT "S SEC NUM = ";SS$
230  PRINT : PRINT "DEPT    =  ";DP$
240  PRINT : PRINT "PAY RATE= ";RP
250  PRINT : INPUT "TO CHANGE ENTER <C>";QU$
260 PF = 1
270 CP =  PEEK (37) - 1: IF QU$ = "C" THEN  GOSUB 620
280  IF PF THEN  GOSUB 430
290  GOTO 160
300  GOSUB 12180: GOSUB 490
310 NA$ = "END RECORD":SS$ = "000-00-0000":DP$ = "00":
     RP = 0:FL = 1: GOSUB 430
320  GOSUB 10210
330  END
340 FL = 0
350  IF SS$ <  > "000-00-0000" OR DP$ <  > "00" OR RP <
      > 0 THEN  RETURN
360  IF NA$ = "END RECORD" THEN FL = 1: RETURN
370  IF NA$ = "END SECTION" THEN FL = 2
380  RETURN
390  PRINT D$;"READ INPUT MASTER"
400  INPUT NA$,SS$,DP$,RP
410  PRINT D$;"IN#0": GOSUB 340
420  RETURN
430  PRINT D$;"WRITE OUTPUT MASTER"
440  PRINT NA$;",";SS$;",";DP$;",";RP
450  PRINT D$;"PR#0"
460 BC = BC +  LEN (NA$) +  LEN (SS$) +  LEN (DP$) +  LEN
     ( STR$ (RP)) + 4
470  IF BC < BL OR FL = 1 THEN  RETURN
480  PRINT D$;"WRITE OUTPUT MASTER": PRINT "END SECTIO
     N,000-00-0000,00,0": GOSUB 10210: GOSUB 10000: RETURN

490  PRINT "ADD MORE RECORDS ";: GOSUB 21000: INPUT "
     ?";QU$
500  IF QU$ <  > "Y" THEN  RETURN
510  HOME
520  GOSUB 580:NA$ = QU$
530  GOSUB 590:SS$ = QU$
540  GOSUB 600:DP$ = QU$
550  GOSUB 610:RP =  VAL (QU$)
560  GOSUB 340: IF FL THEN  PRINT "THIS IS AN ILLEGAL
     RECORD !!!": FOR J = 1 TO 500: NEXT J: PRINT  CHR$
     (7): GOTO 510
```

Fig. 10-3. Program to update a sequential text file.

```
570  GOSUB 430: GOTO 490
580  INPUT "NAME = ";QU$: RETURN
590  INPUT "S SEC NUM = ";QU$: RETURN
600  INPUT "DEPT.  = ";QU$: RETURN
610  INPUT "RATE  = ";QU$: RETURN
620  POKE 37,CP: CALL  - 958: PRINT "ENTER :": PRINT :
      HTAB 7: PRINT "A TO ALTER"
630  HTAB 7: PRINT "I TO INSERT NEW RECORD"
640  HTAB 7: PRINT "D TO DELETE ITEM"
650  HTAB 7: PRINT "C TO ACCEPT ITEM AS IS"
660  PRINT : PRINT : INPUT "ENTER CODE ?";QU$
670  ON ((QU$ = "I") + (QU$ = "A") * 2 + (QU$ = "D") *
     3 + (QU$ = "C") * 4) GOTO 680,790,870,860: GOTO 6
     20
680 L1$ = NA$:L2$ = SS$:L3$ = DP$:LT = RP
690  POKE 37,CP: CALL  - 958
700  GOSUB 580:NA$ = QU$
710  GOSUB 590:SS$ = QU$
720  GOSUB 600:DP$ = QU$
730  GOSUB 610:RP =  VAL (QU$)
740  GOSUB 340: IF FL THEN  PRINT "THIS IS AN ILLEGAL
     RECORD !!!": FOR J = 1 TO 500: NEXT J: PRINT  CHR$
     (7): GOTO 690
750  GOSUB 430: PRINT "INSERT ANOTHER RECORD ";: GOSUB
     21000: INPUT " ?";QU$
760  IF QU$ = "Y" THEN 690
770 NA$ = L1$:SS$ = L2$:DP$ = L3$:RP = LT
780  GOTO 620
790  POKE 37,CP: CALL  - 958: PRINT "ENTER NEW VALUE O
     R 'RETURN'"
800  PRINT "TO LEAVE CURRENT VALUE UNCHANGED"
810  PRINT : GOSUB 580: IF  LEN (QU$) > 0 THEN NA$ = Q
     U$
820  GOSUB 590: IF  LEN (QU$) > 0 THEN SS$ = QU$
830  GOSUB 600: IF  LEN (QU$) > 0 THEN DP$ = QU$
840  GOSUB 610: IF  LEN (QU$) > 0 THEN RP =  VAL (QU$)

850  GOSUB 340: IF FL THEN  PRINT "ILLEGAL RECORD !!!"
     : FOR J = 1 TO 500: NEXT J: PRINT  CHR$ (7): GOTO
     790
860  RETURN
870 PF = 0: RETURN
10000  HOME : PRINT "TAKE OUT CURRENT SECTION AND"
10010  PRINT : PRINT "PUT IN OUTPUT SECTION # ";SO;: INPUT
     " ";QU$
10020  HOME : PRINT D$;"CATALOG,S";SD;",D";DD
10030  PRINT : PRINT "DELETE OLD ";F$;" OFF ": PRINT "
     DESTINATION DRIVE ";: GOSUB 21010: INPUT " ?";QU$
     : IF QU$ = "N" THEN 10100
10040  PRINT D$;"OPEN ";F$;",S";SD;",D";DD
10050  PRINT D$;"READ ";F$
10060  INPUT QU$,ES,QU$,QU$
10070  PRINT D$;"PR#0": PRINT D$;"IN#0"
10080  PRINT "SECTION # ";ES;" HAS DATE OF ";QU$: PRINT
     : PRINT "DO YOU WANT TO DELETE IT ";: GOSUB 21000
     : INPUT " ?";QU$: IF QU$ <  > "Y" THEN 10010
```

Fig. 10-3—*cont.* Program to update a sequential text file.

```
10090  PRINT D$;"UNLOCK ";F$;",S";SD;",D";DD: PRINT D$
    ;"DELETE ";F$;",S";SD;",D";DD
10100  PRINT "DELETE OTHER FILES ";: GOSUB 21000: INPUT
    " ?";QU$
10110  IF QU$ <  > "Y" THEN 10150
10120  PRINT "ENTER THE FILE NAME ?": PRINT : INPUT "N
    AME = ";DF$
10125  PRINT D$;"OPEN ";DF$
10130  PRINT D$;"UNLOCK ";DF$;",S";SD;",D";DD: PRINT D
    $;"DELETE ";DF$;",S";SD;",D";DD
10140  PRINT D$;"CATALOG,S";SD;",D";DD: GOTO 10100
10150  PRINT D$;"OPEN OUTPUT MASTER,S";SD;",D";DD
10160  PRINT D$;"WRITE OUTPUT MASTER"
10170  PRINT "SECTION INDICATOR": PRINT SO: PRINT "DAT
    E RECORD": PRINT CD$
10180 BC = 40
10190 SO = SO + 1
10200  PRINT D$;"PR#0": RETURN
10210  PRINT D$;"CLOSE OUTPUT MASTER": PRINT D$;"UNLOC
    K OUTPUT MASTER,S";SD;",D";DD
10220  PRINT D$;"RENAME OUTPUT MASTER,";F$
10230  PRINT D$;"LOCK ";F$;",S";SD;",D";DD
10240  RETURN
12000  HOME : PRINT "TAKE OUT CURRENT SECTION"
12010  PRINT : PRINT "PUT IN INPUT SECTION # ";SN;: INPUT
    " ";QU$
12020  PRINT D$;"CATALOG,S";SL;",D";DR
12030  PRINT : PRINT "IS CORRECT FILE ON THIS DRIVE ";
    : GOSUB 21010: INPUT " ?";QU$
12040  IF QU$ = "N" THEN 12010
12050  PRINT D$;"OPEN ";F$;",S";SL;",D";DR
12060  PRINT D$;"READ "F$
12070  INPUT QU$,ES,QU$,QU$
12080  PRINT D$;"PR#0": PRINT D$;"IN#0"
12090  IF SN = ES THEN 12120
12100  PRINT "THIS IS SECTION # ";ES: PRINT : PRINT "S
    ECTION # ";SN;" IS NEEDED !": PRINT
12110  PRINT D$;"CLOSE ";F$: GOTO 12010
12120  IF QU$ = LD$ THEN 12150
12130  PRINT "THE FILE DATE DOES NOT MATCH THE": PRINT
    "LAST RUN DATE": PRINT "FILE DATE = ";QU$: PRINT
    "LAST RUN DATE = ";LD$: PRINT
12140  GOTO 12110
12150  PRINT D$;"CLOSE ";F$: PRINT D$;"UNLOCK ";F$: PRINT
    D$;"RENAME ";F$;",INPUT MASTER"
12160  PRINT D$;"OPEN INPUT MASTER": PRINT D$;"READ IN
    PUT MASTER": INPUT QU$,ES,QU$,QU$
12170  PRINT D$;"IN# 0": PRINT D$;" PR# 0":SN = SN + 1
    : RETURN
12180  PRINT D$;"CLOSE INPUT MASTER"
12190  PRINT D$;"UNLOCK INPUT MASTER,S";SL;",D";DR: PRINT
    D$;"RENAME INPUT MASTER,";F$: PRINT D$;"LOCK ";F$
    : RETURN
20000 SL = 6: REM :SLOT # IS 6
20010 DR = 1: REM :DRIVE # IS 1
20020 SD = 6:DD = 2: REM : SLOT AND DRIVE DESTINATION!!!
```

Fig. 10-3—*cont.* Program to update a sequential text file.

```
20030  HOME : VTAB 3
20040  PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
20050  PRINT
20060  PRINT "UPDATED FILE IN SLOT ";SD;" DRIVE ";DD
20070  PRINT
20080  PRINT "CHANGE THESE VALUES ";: GOSUB 21000: INPUT
     " ?";QU$
20090  PRINT
20100  IF QU$ <  > "Y" THEN 20200
20110  INPUT "ENTER SOURCE SLOT (1-7) ?";SL
20120  IF SL < 1 OR SL > 7 THEN 20110
20130  INPUT "ENTER SOURCE DRIVE (1-2) ?";DR
20140  IF DR <  > 1 AND DR <  > 2 THEN 20130
20150  INPUT "ENTER DESTINATION SLOT (1-7) ?";SD
20160  IF SD < 1 OR SD > 7 THEN 20150
20170  INPUT "ENTER DESTINATION DRIVE (1-2) ?";DD
20180  IF DD < 1 OR DD > 2 THEN 20170
20190  IF SL = SD AND DR = DD THEN 20000
20200  PRINT : INPUT " ENTER FILE NAME ?";F$
20210  IF  LEN (F$) = 0 THEN  PRINT : PRINT "*** NAME
     IS INVALID ***": GOTO 20200
20220  FOR J = 1 TO  LEN (F$): IF  ASC ( MID$ (F$,J,1)
     ) < 32 THEN  PRINT : PRINT "*** PLEASE DON'T TRY
     CONTROL CHARA-": PRINT "   CTERS YET !": GOTO 202
     00
20230  NEXT J: PRINT : PRINT "!!! THE NAME IS GOOD"
20240 BL = 300: REM  !!!CHANGE THIS WHEN TESTING IS CO
     MPLETE
20250 SN = 1: REM :SECT 1 OF INPUT
20260 SO = 1: REM :SECT 1 OF OUTPUT
20270  PRINT : PRINT : INPUT "ENTER TODAY'S DATE (MMDD
     YY) ?";QU$
20280  IF  LEN (QU$) <  > 6 THEN 20270
20290  GOSUB 20360: IF DF THEN 20270
20300 CD$ = QU$
20310  PRINT : PRINT : PRINT "ENTER DATE USED FOR LAST
      UPDATE": PRINT : INPUT "(MMDDYY) ?";QU$
20320  IF  LEN (QU$) <  > 6 THEN 20310
20330  GOSUB 20360: IF DF THEN 20310
20340 LD$ = QU$
20350  RETURN
20360 DF = 0:L1 =  VAL ( LEFT$ (QU$,2)):L2 =  VAL ( MID$
     (QU$,3,2)):L3 =  VAL ( RIGHT$ (QU$,2))
20370  IF L2 < 1 OR L2 > 31 THEN 20420
20380  IF L1 = 1 OR L1 = 3 OR L1 = 5 OR L1 = 7 OR L1 =
     8 OR L1 = 10 OR L1 = 12 THEN 20430
20390  IF L2 > 30 THEN 20420
20400  IF L1 <  > 2 THEN 20430
20410  IF L2 <  = 28 + ( INT (L3 / 4) * 4 = L3) - (L3 =
     0) THEN 20430
20420 DF = 1
20430  RETURN
21000  PRINT "Y OR ";: INVERSE : PRINT "N";: NORMAL : RETURN
21010  INVERSE : PRINT "Y";: NORMAL : PRINT " OR N";: RETURN
```

Fig. 10-3—*cont.* Program to update a sequential text file.

days hath September, April, June, and November. All the rest have 31 save February alone, which has 28, except during leap year. Then February has 29.

If the date input is illegal (the date flag remains at zero) on return to line 20290, the program asks for another date to be input.

If the date is legal (DF = 0) on return to line 20290, the program continues.

Line 20300 assigns the date input (QU$) to the current date string (CD$). The date used for the last update is then entered as QU$ and checked. Line 20340 then assigns the last update date (QU$) to the string variable LD$. Later in the program, the date on every section of the INPUT MASTER is compared to the date stored in LD$. The date stored in CD$ is written to the OUTPUT MASTER.

The program returns to line 130, clears the screen, prints to the screen, "PRESS ' RETURN ' WHEN DRIVES ARE READY"?, and stops the program. This gives the user time to place the input file disk in the source drive and the output file disk in the destination drive.

Line 140 is GOSUB 12010. The subroutine at lines 12010 through 12170, allows the user to input the CURRENT MASTER (TEST) file section. The program tells the user to input a specific section number of the file. The disk is CATALOGed, so the user can determine if the correct file is on the disk. The program defaults to line 12070.

```
12070 INPUT QU$, ES, QU$, QU$
```

The information in line 12070 is read from the CURRENT MASTER (TEST) file, and is checked for the section number input. Four pieces of information are contained in the beginning of section word, (1) "SECTION INDICATOR" (QU$), (2) the section number, (ES), (3) "DATE RECORD" (QU$), and (4) the date of the update (QU$). In this case, only the section number (ES), in the second position, and the date of update (QU$), in the fourth position are needed. This is why QU$ can be used as three of the four variables. Memory contains the last variable placed in QU$ location.

If the section number and date of the CURRENT MASTER (TEST) is correct, the CURRENT MASTER (TEST) file is renamed INPUT MASTER, the section of the INPUT MASTER is incremented (SN), and the program returns to line 150. Line 150 is GOSUB 10010.

The subroutine at lines 10010 through 10200 allows the user and program to do several things: (1) enter the OUTPUT MASTER file, (2) catalog the OUTPUT MASTER file, (3) DELETE the dummy file TEST, (4) DELETE other files on the disk, (5) write the slot number and drive number of the

destination disk drive, (6) WRITE the OUTPUT MASTER section and number and date, (7) set the byte count to 40 (BC = 40), (8) increment the section output (SO) number, and (9) return to line 160. Line 160 is GOSUB 390.

The subroutine at lines 390 through 420 reads a record from the INPUT MASTER file, and jumps to the subroutine at lines 340 through 380.

The subroutine that begins at line 340 assigns the flag (FL = 0) a value of zero. The subroutine then checks to determine if the record read is an employee record (if it is, leaves FL = 0), a special end of file record (if it is, FL = 1), or the end of section record (if it is, FL = 2). The program returns to line 170.

The two program statements at lines 170 and 180 are designed to select one of three options: (1) FL = 0, (2) FL = 1, or (3) FL = 2.

When the flag value is set to two (FL = 2), an end of section word has been read. This causes the program to default through lines 170 and 180 to line 190 which is, GOSUB 12180 : GOSUB 12000 : GOTO 160.

The subroutine at 12180 CLOSEs the INPUT MASTER file, UNLOCKs INPUT MASTER, RENAMEs INPUT MASTER, TEST, LOCKs TEST and returns to GOSUB 12000.

The subroutine that begins at line 12000 instructs the user to "TAKE OUT CURRENT SECTION—PUT IN INPUT SECTION #2", CATALOG S6, D1. The program then asks, "IS CORRECT FILE ON THIS DRIVE?". If the user types **Y** for yes, the program OPENs TEST, S6,D1, READs TEST, and reads the beginning section word for section #2. If the correct section and date were input, the program CLOSEs TEST, RENAMEs TEST, INPUT MASTER, OPENs and READs INPUT MASTER.

In line 12170 the section number (SN) input is incremented and the program returns to GOTO 160, which is GOSUB 390.

The subroutine at 390 reads the next record from the INPUT MASTER, and calls the subroutine at lines 340 through 380.

The subroutine at 340 sets the flag to zero (FL = 0). If the record just read from the INPUT MASTER is an end of file record, the flag is set to one (1), the subroutines CLOSE INPUT MASTER, and RENAME it TEST. The user is asked if there are more records to be added. If the response is **N**, an end of file word is placed on the OUTPUT MASTER, the OUTPUT MASTER is CLOSEd and RENAMEd TEST.

```
400 INPUT NA$, SS$, DP$, HP RETURN
460 BC=BC + LEN(NA$) + LEN(SS$) + LEN(DP$) + LEN(STR$(HP)) + 4
```

Line 460 sums the total number of bytes used in the record. The record

length is computed by adding the previous byte count (BC) to the name string, plus the social security string, plus the department string, plus the rate of pay string. In line 400 there are three commas and an invisible RETURN character that must be added to the byte count. The plus four (+4) at the end of line 360 adds in the three commas and the invisible RETURN character.

```
180 is IF FL = 0 THEN 200
```

The lines at 200 through 270 allow the user to make any necessary changes to maintain the employee records.

Lines 200 through 270 in Fig. 10-3 are similar to lines 320 through 380 in Fig. 8-5.

```
280 IF PF THEN GOSUB 430
```

Using the print flag (PF) variable is a method to delete the record. When the print flag equals zero (PF = 0), the record is deleted by not writing the record to the OUTPUT MASTER. If the print flag variable is assigned the value of one (PF = 1), the record is written to the OUTPUT MASTER file.

```
170 IF FL = 1 THEN 300
```

Line 300 is GOSUB 12180. This subroutine closes out the old INPUT MASTER and RENAMEs it (new) INPUT MASTER with a new section number and date. The program then goes to the subroutine at line 490, to ask the user if more records are to be input. If no more records are to be input, the program returns to line 310 to place the end of file record on the file, WRITE OUTPUT MASTER, print the end of record word, close and lock the OUTPUT MASTER. The program also unlocks the OUTPUT MASTER, RENAMEs the OUTPUT MASTER, TEST, and locks the TEST file.

The subroutines 490 through 570, and 580 through 610 in Fig. 10-3, are similar to the subroutines 600 through 680, and 690 through 720 in Fig. 8-5.

Lines 750, 10100, and 20080 call a subroutine at line 21000.

```
21000 PRINT "Y OR ";:INVERSE: PRINT "N";:NORMAL: RETURN
```

This is the "Y" or "N" print on the screen in response to the interactive question, "INSERT ANOTHER RECORD?" The "N" is in the inverse mode which shows a black "N" in a white background (on a black and white TV). The inverse mode alerts the user that this is a special "N" and by pressing RETURN the "NO" response is selected. This saves a step by not having to type N, and then press RETURN.

Line 10030 calls the subroutine at line 21010.

```
21010 INVERSE:PRINT "Y";:NORMAL:PRINT " OR N";:RETURN
```

This subroutine places the "Y" in the inverse mode. When the interactive question in line 10030, "DELETE OLD ";F$;" OFF":PRINT "DESTINATION DRIVE":, is printed on the screen, pressing RETURN selects the yes option. This saves a step by not having to type in Y, and then press RETURN.

Fig. 10-4 is the first update of the UPDATE program in Fig. 10-3. The dummy file TEST, Fig. 10-2, was placed on disk when the CREATE program, Fig. 10-1, was RUN. The file TEST contains no data, but consists of the beginning of section word, and the end of file word. The file TEST is LOCKed, as the UPDATE program begins.

The first printed line in Fig. 10-4 is MON C, I, O which means that all monitor commands, input, and output are printed on the hardcopy, for the reader's clarification. If NOMON C, I, O turns off the MONITOR, many of the printed lines would not be output.

The unit accessed (source drive) is Slot 6, Drive 1. The update file (destination drive) is Slot 6, Drive 2. These are the default drive values and their values are not changed. The program asks the question, "ENTER FILE NAME?", and the user responds with TEST. The program outputs, "!!! THE NAME IS GOOD".

The program asks, "ENTER TODAY'S DATE (MMDDYY)", and the user types in 122081. The program then instructs the user, "ENTER DATE USED FOR LAST UPDATE (MMDDYY)?", and the user responds with 121581.

The program instructs the user, "PRESS 'RETURN' WHEN DRIVES ARE READY!", and the program halts. After the CURRENT (INPUT) MASTER file is placed in the source drive, S6, D1, and the OUTPUT MASTER is placed in the destination drive, S6, D2, the user presses the RETURN key.

The program prints out, "PUT IN INPUT SECTION #1", the program does a catalog on the disk in slot #6, drive #1.

The program asks, "IS CORRECT FILE ON THIS DISK Y OR N?", and the user types Y for yes and presses RETURN.

The file TEST is opened, read and "SECTION INDICATOR", SECTION NUMBER 1, "DATE RECORD", and the date of the last update (121581) are placed on the screen. Line

```
12080 PRINT D$;"PR#0":PRINT D$;"IN#0"
```

causes input to be taken from the keyboard and output to go to the screen.

```
]MON C,I,O

]RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

UPDATED FILE IN SLOT 6 DRIVE 2

CHANGE THESE VALUES Y OR N ?


 ENTER FILE NAME ?TEST

!!! THE NAME IS GOOD


ENTER TODAY'S DATE (MMDDYY) ?122081


ENTER DATE USED FOR LAST UPDATE

(MMDDYY) ?121581
PRESS 'RETURN' WHEN DRIVES ARE READY!

PUT IN INPUT SECTION # 1
CATALOG,S6,D1

DISK VOLUME 254

*A 002 DISKINIT
*T 002 TEST


IS CORRECT FILE ON THIS DRIVE Y OR N ?Y
OPEN TEST,S6,D1
READ TEST
?SECTION INDICATOR
??1
??DATE RECORD
??121581
PR#1
IN#0
CLOSE TEST
UNLOCK TEST
RENAME TEST,INPUT MASTER
OPEN INPUT MASTER
READ INPUT MASTER
?SECTION INDICATOR
??1
```

Fig. 10-4. First update of Fig. 10-3.

```
??DATE RECORD
??121581
IN# 0
 PR# 1

PUT IN OUTPUT SECTION # 1


CATALOG,S6,D2

DISK VOLUME 254

*A 002 DISKINIT


DELETE OLD TEST OFF
DESTINATION DRIVE Y OR N ?N
DELETE OTHER FILES Y OR N ?N
OPEN OUTPUT MASTER,S6,D2
WRITE OUTPUT MASTER
SECTIQN INDICATOR
1
DATE RECORD
122081
PR#1
READ INPUT MASTER
?END RECORD
??000-00-0000
??00
??00.00
IN#0
CLOSE INPUT MASTER
UNLOCK INPUT MASTER,S6,Dl
RENAME INPUT MASTER,TEST
LOCK TEST
ADD MORE RECORDS Y OR N ?Y
NAME = JOHN J. ALDRIGE
S SEC NUM = 000-00-0000
DEPT.  = SHIPPING
RATE  = 8.00
WRITE OUTPUT MASTER
JOHN J. ALDRIGE,000-00-0000,SHIPPING,8
PR#1
ADD MORE RECORDS Y OR N ?Y
NAME = WILLIAM Z. BASS
S SEC NUM = 000-00-0000
DEPT.  = SALES
```

Fig. 10-4—*cont.* First update of Fig. 10-3.

```
RATE  = 20.00
WRITE OUTPUT MASTER
WILLIAM Z. BASS,000-00-0000,SALES,20
PR#1
ADD MORE RECORDS Y OR N ?Y
NAME = ANTHONTY X. BECKER
S SEC NUM = 000-00-0000
DEPT.  = SALES
RATE  = 15.00
WRITE OUTPUT MASTER
ANTHONTY X. BECKER,000-00-0000,SALES,15
PR#1
ADD MORE RECORDS Y OR N ? Y
NAME = HERMAN A. CAMPBELL
S SEC NUM = 000-00-0000
DEPT.  = ACCOUNTING
RATE  = 20.50
WRITE OUTPUT MASTER
HERMAN A. CAMPBELL,000-00-0000,ACCOUNTING,20.5
PR#1
ADD MORE RECORDS Y OR N ?Y
NAME = AL O. CUNNINGHAM
S SEC NUM = 000-00-0000
DEPT.  = SALES
RATE  = 10.00
WRITE OUTPUT MASTER
AL O. CUNNINGHAM,000-00-0000,SALES,10
PR#1
ADD MORE RECORDS Y OR N ?Y
NAME = JOHN Y. FAULK
S SEC NUM = 000-00-0000
DEPT.  = SALES
RATE  = 10.00
WRITE OUTPUT MASTER
JOHN Y. FAULK,000-00-0000,SALES,10
PR#1
ADD MORE RECORDS Y OR N ?Y
NAME = BILLY F. GARRETT JR.
S SEC NUM = 000-00-0000
DEPT.  = SALES
RATE  = 10.00
WRITE OUTPUT MASTER
BILLY F. GARRETT JR.,000-00-0000,SALES,10
PR#1
WRITE OUTPUT MASTER
END SECTION,000-00-0000,00,0
CLOSE OUTPUT MASTER
UNLOCK OUTPUT MASTER,S6,D2
```

Fig. 10-4—*cont.* First update of Fig. 10-3.

```
RENAME OUTPUT MASTER,TEST
LOCK TEST,S6,D2
TAKE OUT CURRENT SECTION AND

PUT IN OUTPUT SECTION # 2
CATALOG,S6,D2

DISK VOLUME 254

*A 002 DISKINIT

DELETE OLD TEST OFF
DESTINATION DRIVE Y OR N ? N
DELETE OTHER FILES Y OR N ?N
OPEN OUTPUT MASTER,S6,D2
WRITE OUTPUT MASTER
SECTION INDICATOR
2
DATE RECORD
122081
PR#1
ADD MORE RECORDS Y OR N ?Y
NAME = LEUWANDA O. HAMILTON
S SEC NUM = 000-00-0000
DEPT.  = VICE-PRESIDENT
RATE  = 50.00
WRITE OUTPUT MASTER
LEUWANDA O. HAMILTON,000-00-0000,VICE-PRESIDENT,50
PR#1
ADD MORE RECORDS Y OR N ?Y
NAME = LEON B. JENNINGS
S SEC NUM = 000-00-0000
DEPT.  = SALES
RATE  = 10.00
WRITE OUTPUT MASTER
LEON B. JENNINGS,000-00-0000,SALES,10
PR#1
ADD MORE RECORDS Y OR N ?N
WRITE OUTPUT MASTER
END RECORD,000-00-0000,00,0
PR#1
CLOSE OUTPUT MASTER
UNLOCK OUTPUT MASTER,S6,D2
RENAME OUTPUT MASTER,TEST
LOCK TEST,S6,D2

]

 PR#0
```

Fig. 10-4—*cont.* First update of Fig. 10-3.

PRINT D$;"PR#0" changes the data flow so that it goes to the screen instead of to the disk, Table 1-3.

PRINT D$;"IN#0" is related to the INPUT command, so the input comes from the keyboard, Table 1-3.

The program CLOSEs TEST, RENAMEs TEST, INPUT MASTER, and READs INPUT MASTER. This action makes the change from TEST to INPUT MASTER so the update routine can proceed according to its normal cycle, as discussed on the first page of Chapter 10.

The program then prints, "PUT IN OUTPUT SECTION #1". The OUTPUT MASTER IS PLACED IN THE DESTINATION DRIVE IN SLOT #6, DRIVE #2. This disk is cataloged. The program asks, "DELETE OLD TEST OFF DESTINATION DRIVE Y OR N?". The user types N for no. The program asks, "DELETE OTHER FILES Y OR N?", and the user types N for no.

The program OPENs OUTPUT MASTER, S6, D2, WRITEs OUTPUT MASTER, and the beginning of section word.

The program READs INPUT MASTER, and places the end of file word on the hardcopy.

The program CLOSEs INPUT MASTER, UNLOCKs INPUT MASTER, S6, D1, RENAMEs INPUT MASTER, TEST, and LOCKs TEST.

The file is renamed TEST so the sequence and the program will run properly. This seems like an extra routine, but TEST is used as a return point so the file can maintain continuity.

The program asks, "DO YOU WANT TO DELETE IT?" (Fig. 10-5), and the user types Y for yes. The program UNLOCKs TEST, S6, D2, and DELETEs TEST S6, D2. The program then asks, "DELETE OTHER FILES Y OR N?". The user types Y for yes.

The program requests, "ENTER THE FILE NAME?". The file name WORTHLESS is entered. This is an example of how the program handles files that are not on the disk, or the user misspells the name of a file on the disk. In order to prevent program difficulties, a file named WORTHLESS is OPENed, UNLOCKed, and DELETEd. If a valid file name had been typed, the program would have deleted it correctly.

The disk is cataloged and the program asks, "DELETE OTHER FILES?", and the user types N for no. The program OPENs OUTPUT MASTER, S6, D2, WRITEs OUTPUT MASTER, and places the current beginning of sector word (122181) on the hardcopy.

The first employee record on the INPUT MASTER file is READ. This is the record of John J. Aldrige. The program requests, "TO CHANGE ENTER <C>. The user does not need to change this record. RETURN is

```
 MON C,I,O

]RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

UPDATED FILE IN SLOT 6 DRIVE 2

CHANGE THESE VALUES Y OR N ?N


 ENTER FILE NAME ?TEST

!!! THE NAME IS GOOD


ENTER TODAY'S DATE (MMDDYY) ?122181


ENTER DATE USED FOR LAST UPDATE

(MMDDYY) ?122081
PRESS 'RETURN' WHEN DRIVES ARE READY!

PUT IN INPUT SECTION # 1
CATALOG,S6,D1

DISK VOLUME 254

*A 002 DISKINIT
*T 003 TEST


IS CORRECT FILE ON THIS DRIVE Y OR N ?Y
OPEN TEST,S6,D1
READ TEST
?SECTION INDICATOR
??1
??DATE RECORD
??122081
PR#1
IN#0
CLOSE TEST
UNLOCK TEST
RENAME TEST,INPUT MASTER
OPEN INPUT MASTER
READ INPUT MASTER
?SECTION INDICATOR
??1
```

Fig. 10-5. Second update of Fig. 10-3.

```
??DATE RECORD
??122081
IN# 0
 PR# 1

PUT IN OUTPUT SECTION # 1
CATALOG,S6,D2

DISK VOLUME 254

*A 002 DISKINIT
*T 002 TEST


DELETE OLD TEST OFF
DESTINATION DRIVE Y OR N ?Y
OPEN TEST,S6,D2
READ TEST
?SECTION INDICATOR
??1
??DATE RECORD
??121581
PR#1
IN#0
SECTION # 1 HAS DATE OF 121581

DO YOU WANT TO DELETE IT Y OR N ?Y
UNLOCK TEST,S6,D2
DELETE TEST,S6,D2
DELETE OTHER FILES Y OR N ?Y
ENTER THE FILE NAME ?

NAME = WORTHLESS
OPEN WORTHLESS
UNLOCK WORTHLESS,S6,D2
DELETE WORTHLESS,S6,D2
CATALOG,S6,D2

DISK VOLUME 254

*A 002 DISKINIT

DELETE OTHER FILES Y OR N ?N
OPEN OUTPUT MASTER,S6,D2
WRITE OUTPUT MASTER
SECTION INDICATOR
1
DATE RECORD
```

Fig. 10-5—*cont.* Second update of Fig. 10-3.

```
122181
PR#1
READ INPUT MASTER
?JOHN J. ALDRIGE,000-00-0000,SHIPPING,8
IN#0
NAME =    JOHN J. ALDRIGE

S SEC NUM = 000-00-0000

DEPT   =  SHIPPING

PAY RATE= 8

TO CHANGE ENTER <C>
WRITE OUTPUT MASTER
JOHN J. ALDRIGE,000-00-0000,SHIPPING,8
PR#1
READ INPUT MASTER
?WILLIAM Z. BASS,000-00-0000,SALES,20
IN#0
NAME =    WILLIAM Z. BASS

S SEC NUM = 000-00-0000

DEPT   =  SALES

PAY RATE= 20

TO CHANGE ENTER <C>C
ENTER :

      A TO ALTER
      I TO INSERT NEW RECORD
      D TO DELETE ITEM
      C TO ACCEPT ITEM AS IS


ENTER CODE ?I
NAME = HYMAN T. ALSTON
S SEC NUM = 000-00-0000
DEPT.  = SHIPPING
RATE   = 8.00
WRITE OUTPUT MASTER
HYMAN T. ALSTON,000-00-0000,SHIPPING,8
PR#1
INSERT ANOTHER RECORD Y OR N ?N
```

Fig. 10-5—*cont.* Second update of Fig. 10-3.

```
ENTER :

      A TO ALTER
      I TO INSERT NEW RECORD
      D TO DELETE ITEM
      C TO ACCEPT ITEM AS IS


ENTER CODE ?C
WRITE OUTPUT MASTER
WILLIAM Z. BASS,000-00-0000,SALES,20
PR#1
READ INPUT MASTER
?ANTHONTY X. BECKER,000-00-0000,SALES,15
IN#0
NAME =     ANTHONTY X. BECKER

S SEC NUM = 000-00-0000

DEPT    =  SALES

PAY RATE= 15

TO CHANGE ENTER <C> C
ENTER :

      A TO ALTER
      I TO INSERT NEW RECORD
      D TO DELETE ITEM
      C TO ACCEPT ITEM AS IS


ENTER CODE ?D
READ INPUT MASTER
?HERMAN A. CAMPBELL,000-00-0000,ACCOUNTING,20.5
IN#0
NAME =     HERMAN A. CAMPBELL

S SEC NUM = 000-00-0000

DEPT    =  ACCOUNTING

PAY RATE= 20.5

TO CHANGE ENTER <C>
WRITE OUTPUT MASTER
HERMAN A. CAMPBELL,000-00-0000,ACCOUNTING,20.5
PR#1
```

Fig. 10-5—*cont.* Second update of Fig. 10-3.

```
READ INPUT MASTER
?AL O. CUNNINGHAM,000-00-0000,SALES,10
IN#0
NAME =     AL O. CUNNINGHAM

S SEC NUM = 000-00-0000

DEPT    =  SALES

PAY RATE= 10

TO CHANGE ENTER <C>C
ENTER :

       A TO ALTER
       I TO INSERT NEW RECORD
       D TO DELETE ITEM
       C TO ACCEPT ITEM AS IS

ENTER CODE ? A
ENTER NEW VALUE OR 'RETURN'
TO LEAVE CURRENT VALUE UNCHANGED

NAME = ALBERT O. CUNNINGHAM
S SEC NUM =
DEPT.  =
RATE   =
WRITE OUTPUT MASTER
ALBERT O. CUNNINGHAM,000-00-0000,SALES,10
PR#1
READ INPUT MASTER
?JOHN Y. FAULK,000-00-0000,SALES,10
IN#0
NAME =     JOHN Y. FAULK

S SEC NUM = 000-00-0000

DEPT    =  SALES

PAY RATE= 10

TO CHANGE ENTER <C>C
ENTER :

       A TO ALTER
       I TO INSERT NEW RECORD
       D TO DELETE ITEM
       C TO ACCEPT ITEM AS IS
```

Fig. 10-5—*cont.* Second update of Fig. 10-3.

```
ENTER CODE ?I
NAME = BILL N. FALLS
S SEC NUM = 000-00-0000
DEPT.  = ACCOUNTING
RATE  = 10.00
WRITE OUTPUT MASTER
BILL N. FALLS,000-00-0000,ACCOUNTING,10
PR#1
INSERT ANOTHER RECORD Y OR N ?N
ENTER :

      A TO ALTER
      I TO INSERT NEW RECORD
      D TO DELETE ITEM
      C TO ACCEPT ITEM AS IS


ENTER CODE ? C
WRITE OUTPUT MASTER
JOHN Y. FAULK,000-00-0000,SALES,10
PR#1
WRITE OUTPUT MASTER
END SECTION,000-00-0000,00,0
CLOSE OUTPUT MASTER
UNLOCK OUTPUT MASTER,S6,D2
RENAME OUTPUT MASTER,TEST
LOCK TEST,S6,D2
TAKE OUT CURRENT SECTION AND

PUT IN OUTPUT SECTION # 2
CATALOG,S6,D2

DISK VOLUME 254

*A 002 DISKINIT


DELETE OLD TEST OFF
DESTINATION DRIVE Y OR N ?N
DELETE OTHER FILES Y OR N ?N
OPEN OUTPUT MASTER,S6,D2
WRITE OUTPUT MASTER
SECTION INDICATOR
2
DATE RECORD
122181
PR#1
READ INPUT MASTER
```

Fig. 10-5—*cont.* Second update of Fig. 10-3.

```
?BILLY F. GARRETT JR.,000-00-0000,SALES,10
IN#0
NAME =     BILLY F. GARRETT JR.

S SEC NUM = 000-00-0000

DEPT    =  SALES

PAY RATE= 10

TO CHANGE ENTER <C>
WRITE OUTPUT MASTER
BILLY F. GARRETT JR.,000-00-0000,SALES,10
PR#1
READ INPUT MASTER
?END SECTION,000-00-0000,00,0
IN#0
CLOSE INPUT MASTER
UNLOCK INPUT MASTER,S6,D1
RENAME INPUT MASTER,TEST
LOCK TEST
TAKE OUT CURRENT SECTION

PUT IN INPUT SECTION # 2
CATALOG,S6,D1

DISK VOLUME 254

*A 002 DISKINIT
*T 002 TEST

IS CORRECT FILE ON THIS DISK Y OR N ?Y
OPEN TEST,S6,D1
READ TEST
?SECTION INDICATOR
??2
??DATE RECORD
??122081
PR#1
IN#0
CLOSE TEST
UNLOCK TEST
RENAME TEST,INPUT MASTER
OPEN INPUT MASTER
READ INPUT MASTER
?SECTION INDICATOR
??2
??DATE RECORD
```

Fig. 10-5—*cont.* Second update of Fig. 10-3.

```
??122081
IN# 0
 PR# 1
READ INPUT MASTER
?LEUWANDA O. HAMILTON,000-00-0000,VICE-PRESIDENT,50
IN#0
NAME =     LEUWANDA O. HAMILTON

S SEC NUM = 000-00-0000

DEPT    =  VICE-PRESIDENT

PAY RATE= 50

TO CHANGE ENTER <C>
WRITE OUTPUT MASTER
LEUWANDA O. HAMILTON,000-00-0000,VICE-PRESIDENT,50
PR#1
READ INPUT MASTER
?LEON B. JENNINGS,000-00-0000,SALES,10
IN#0
NAME =     LEON B. JENNINGS

S SEC NUM = 000-00-0000

DEPT    =  SALES

PAY RATE= 10

TO CHANGE ENTER <C>
WRITE OUTPUT MASTER
LEON B. JENNINGS,000-00-0000,SALES,10
PR#1
READ INPUT MASTER
?END RECORD,000-00-0000,00,0
IN#0
CLOSE INPUT MASTER
UNLOCK INPUT MASTER,S6,D1
RENAME INPUT MASTER,TEST
LOCK TEST
ADD MORE RECORDS Y OR N ?N
WRITE OUTPUT MASTER
END RECORD,000-00-0000,00,0
PR#1
CLOSE OUTPUT MASTER
UNLOCK OUTPUT MASTER,S6,D2
RENAME OUTPUT MASTER,TEST
LOCK TEST,S6,D2
]PR#0
```

Fig. 10-5—*cont.* Second update of Fig. 10-3.

pressed and John J. Aldrige's personnel record is written to the OUTPUT MASTER file.

The program READs INPUT MASTER to place the next employee record on the screen. The record of Hyman T. Alston is to be placed in alphabetical order before the record of William Z. Bass.

The program requests, "TO CHANGE ENTER <C>. The user types C for change, and the CHANGE menu appears on the screen. The menu has four options.

```
A TO ALTER
I TO INSERT
D TO DELETE
C TO ACCEPT ITEM AS IS
```

Since a record is to be inserted, I is typed and the record is input.

```
ENTER CODE? I
NAME = HYMAN T. ALSTON
S SEC NUM = 000-00-0000
DEPT. = SHIPPING
RATE = 8.00
```

The Hyman T. Alston record is written to the OUTPUT MASTER file. The program asks, "INSERT ANOTHER RECORD Y OR N?", and the user types N for no. The menu item is then selected, "C TO ACCEPT ITEM AS IS", and the record of William Z. Bass is written to the OUTPUT MASTER.

The file of Anthony X. Becker is then read from the INPUT MASTER, and this file is to be deleted from the OUTPUT MASTER file. "D TO DELETE ITEM" is selected from the menu, and the next record is read from the INPUT MASTER and placed on the screen for user inspection. The next record is that of Herman A. Campbell, and no changes are to be made. RETURN is pressed. The record of Herman A. Campbell is written to the OUTPUT MASTER.

The next record is read from the INPUT MASTER. The first name of Al O. Cunningham is to be changed to Albert. "A TO ALTER" is selected from the menu and Albert O. Cunningham is typed after the NAME = indicator. No other changes are made to the record, so RETURN is pressed after S SEC NUM = , DEPT. = , and RATE = . The altered record of Albert O. Cunningham is written to the OUTPUT MASTER.

The INPUT MASTER then reads the record of John Y. Faulk. The record of Bill N. Falls is to be inserted before the record of John Y. Faulk, so "I

TO INSERT" is selected from the menu. The four items in the record of Bill N. Falls are input and written to the OUTPUT MASTER.

The record of John Y. Faulk fills section one (1) of the OUTPUT MASTER, so the end of section word is written to the OUTPUT MASTER file, and the OUTPUT MASTER is CLOSEd, UNLOCKed, and RENAMEd TEST. TEST is locked in S6, D2.

The program requests, "PUT IN OUTPUT SECTION #2". The user complies with the request and places a formatted disk in the destination drive, and presses RETURN. The program CATALOGs the disk in drive in slot #6, drive #2.

The program asks, "DELETE OLD TEST OFF DESTINATION DRIVE Y OR N?". The response is N for no. The program asks, "DELETE OTHER FILES Y OR N?", and again the response is N for no.

The program OPENs OUTPUT MASTER, S6, D2, WRITEs OUTPUT MASTER, and writes the new beginning of section word, including the date (122181).

The INPUT MASTER reads the record of Billy F. Garrett Jr. No changes are to be made, RETURN is pressed, and the record of Billy F. Garrett Jr. is written to the OUTPUT MASTER.

The INPUT MASTER reads the next record, which is the end of section word. The end of section record on INPUT SECTION #1 causes the INPUT MASTER, S6, D1 to be CLOSEd, and RENAMEd TEST. The program requests the user to, "TAKE OUT CURRENT SECTION—PUT IN INPUT SECTION #2. The user complied with the request. When INPUT SECTION #2 is placed in the source drive, RETURN is pressed. The disk in the drive in slot #6, drive #1, is CATALOGed.

The program asks, "IS CORRECT FILE ON THIS DISK?" and the user types Y for yes. The file TEST is OPENed and READ, and the beginning of sector word (#2) is placed on the hardcopy, including the date (122081).

The program CLOSEs TEST, UNLOCKs TEST, RENAMEs TEST, INPUT MASTER, OPENs INPUT MASTER, and READs INPUT MASTER, and places the beginning of section (#2) word on the hardcopy.

The first record read from the INPUT MASTER is that of Leuwanda O. Hamilton. No changes are to be made to this record, RETURN is pressed. The Hamilton record is written to the OUTPUT MASTER, and the record of Leon B. Jennings is read from the INPUT MASTER. No changes are to be made to this record, so RETURN is pressed. The Jennings record is written to the OUTPUT MASTER.

The INPUT MASTER reads the next record, which is the end of file

```
100  REM :PROGRAM TO READ A      SEQUENTIAL FILE
110 D$ =  CHR$ (4)
120 SL = 6: REM :SLOT IS #6
130 DR = 1: REM :DRIVE IS #1
140  HOME : VTAB 3
150  PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
160  PRINT
170  INPUT "CHANGE THESE VALUES(Y OR N)? ";QU$
180  PRINT
190  IF QU$ <  > "Y" THEN 240
200  INPUT "ENTER SLOT (1-7) ?";SL
210  IF SL < 1 OR SL > 7 THEN 200
220  INPUT "ENTER DRIVE (1-2) ?";DR
230  IF DR <  > 1 AND DR <  > 2 THEN 220
240  PRINT : INPUT " ENTER FILE NAME ?";F$
250  IF  LEN (F$) = 0 THEN  PRINT : PRINT "*** NAME IS
      INVALID ***": GOTO 240
260  FOR J = 1 TO  LEN (F$): IF  ASC ( MID$ (F$,J,1)) <
     32 THEN  PRINT : PRINT "*** PLEASE DON'T TRY CONT
     ROL CHARA-": PRINT "CTERS YET !": GOTO 240
270  NEXT J: PRINT : PRINT "!!! THE NAME IS GOOD"
290  HOME : PRINT "ENTER VALUE FOR SELECTION ELSE"
300  PRINT : PRINT "PRESS 'RETURN' TO LIST ALL ENTRIES
     "
310  PRINT : INPUT "ENTER NAME TOKEN ?";TN$
315 LN =  LEN (TN$)
320  PRINT : INPUT "ENTER SS NUM TOKEN ?";TS$
325 LS =  LEN (TS$)
330  PRINT : INPUT "ENTER DEPT TOKEN ?";DT$
335 LT =  LEN (DT$)
340  HOME : PRINT : PRINT "ENTER RANGE OF PAY RATE<R>"

350  PRINT : PRINT "SELECT ABOVE MINIMUM <M>"
360  PRINT : PRINT "SELECT BELOW MAXIMUM<L>"
370  PRINT : PRINT "ACCEPT ALL RATES<A>"
380  PRINT : INPUT "ENTER SELECTION ?";QU$
390  ON ((QU$ = "M") + (QU$ = "L") * 2 + (QU$ = "R") *
     3 + (QU$ = "A") * 4) GOTO 400,430,460,490: GOTO 3
     40
400  HOME : PRINT : INPUT "ENTER MINIMUM RATE ?";LR
410 UR = 5.0E + 9: IF LR < 0 THEN 400
420  GOTO 500
430  HOME : PRINT : INPUT "ENTER UPPER LIMIT ?";UR
440 LR = 0: IF LR > UR THEN 430
450  GOTO 500
460  HOME : INPUT "ENTER BASE FIGURE ?";LR: INPUT "ENT
     ER MAXIMUM FIGURE ?";UR
470  IF LR < 0 OR UR < 0 OR UR < LR THEN 460
480  GOTO 500
490 LR = 0:UR = 5E + 9
500  HOME :SN = 1: GOTO 520
510  PRINT D$;"PR#0": PRINT D$;"IN#0": PRINT : PRINT :
      PRINT : PRINT "TAKE OUT CURRENT SECTION !": PRINT
520  GOSUB 12010
530  GOSUB 900: ON FL + 1 GOTO 535,760,510
535 L1 =  LEN (NA$):L2 =  LEN (SS$):L3 =  LEN (DP$)
```

Fig. 10-6. Program to read (list) all, or parts of, a sequential text file.

```
540  IF LN = 0 THEN 590
550  IF L1 < LN THEN 750
560 L = L1 - LN + 1: FOR J = 1 TO L
570  IF TN$ =  MID$ (NA$,J,LN) THEN 590
580  NEXT J: GOTO 750
590  IF LS = 0 THEN 640
600  IF L2 < LS THEN 750
610 L = L2 - LS + 1: FOR J = 1 TO L
620  IF TS$ =  MID$ (SS$,J,LS) THEN 640
630  NEXT J: GOTO 750
640  IF LT = 0 THEN 690
650  IF L3 < LT THEN 750
660 L = L3 - LT + 1: FOR J = 1 TO L
670  IF DT$ =  MID$ (DP$,J,LT) THEN 690
680  NEXT J: GOTO 750
690  IF RP < LR OR RP > UR THEN 750
700  PRINT "NAME =     ";NA$
710  PRINT "S SEC # = ";SS$
720  PRINT "DEPT     = ";DP$
730  PRINT "PAY RATE= ";RP
740  PRINT : PRINT "----------": PRINT
750  GOTO 530
760  PRINT D$;"IN#0"
770  END
800  INPUT QU$,ES,QU$,QU$
810  RETURN
900  INPUT NA$,SS$,DP$,RP
910 FL = 0
920  IF SS$ <  > "000-00-0000" OR DP$ <  > "00" OR RP <
       > 0 THEN  RETURN
930  IF NA$ = "END RECORD" THEN 980
940  IF NA$ = "END SECTION" THEN 990
950  RETURN
980 FL = 1: RETURN
990 FL = 2: RETURN
12000  PRINT D$;"PR#0": PRINT D$;"IN#0"
12010  PRINT : PRINT "PUT IN INPUT SECTION # ";SN;: INPUT
     " ";QU$
12020  PRINT D$;"CATALOG,S";SL;",D";DR
12030  PRINT : PRINT "IS CORRECT FILE ON THIS DRIVE ";
     : GOSUB 21010: INPUT " ?";QU$
12040  IF QU$ = "N" THEN 12010
12050  PRINT D$;"OPEN ";F$;",S";SL;",D";DR
12060  PRINT D$;"READ "F$
12070  INPUT QU$,ES,QU$,QU$
12080  PRINT D$;"PR#0": PRINT D$;"IN#0"
12090  IF SN = ES THEN 12120
12100  PRINT "THIS IS SECTION # ";ES: PRINT : PRINT "S
     ECTION # ";SN;" IS NEEDED !": PRINT
12110  PRINT D$;"CLOSE ";F$: GOTO 12010
12120  PRINT : PRINT "DATE OF FILE IS : ";QU$
12130  PRINT D$;"PR# 0": PRINT D$;"READ ";F$:SN = SN +
     1: RETURN
21000  PRINT "Y OR ";: INVERSE : PRINT "N";: NORMAL : RETURN
21010  INVERSE : PRINT "Y";: NORMAL : PRINT " OR N";: RETURN
```

Fig. 10-6—*cont.* Program to read (list) all, or parts of, a sequential text file.

```
12010  PRINT : PRINT "PUT IN INPUT SECTION # ";SN;: INPUT
     " ";QU$
12020  PRINT D$;"CATALOG,S";SL;",D";DR
12030  PRINT : PRINT "IS CORRECT FILE ON THIS DRIVE ";
     : GOSUB 21010: INPUT " ?";QU$
12040  IF QU$ = "N" THEN 12010
12050  PRINT D$;"OPEN ";F$;",S";SL;",D";DR
12060  PRINT D$;"READ "F$
12070  INPUT QU$,ES,QU$,QU$
12080  PRINT D$;"PR#0": PRINT D$;"IN#0"
12090  IF SN = ES THEN 12120
12100  PRINT "THIS IS SECTION # ";ES: PRINT : PRINT "S
     ECTION # ";SN;" IS NEEDED !": PRINT
12110  PRINT D$;"CLOSE ";F$: GOTO 12010
12120  IF QU$ = LD$ THEN 12150
12130  PRINT "THE FILE DATE DOES NOT MATCH THE": PRINT
     "LAST RUN DATE": PRINT "FILE DATE = ";QU$: PRINT
     "LAST RUN DATE = ";LD$: PRINT
12140  GOTO 12110
12150  PRINT D$;"IN# 0": PRINT D$;"PR# 0":SN = SN + 1:
     RETURN
21000  PRINT "Y OR ";: INVERSE : PRINT "N";: NORMAL : RETURN
21010  INVERSE : PRINT "Y";: NORMAL : PRINT " OR N";: RETURN
```

Fig. 10-6—*cont.* Program to read (list) all, or parts of, a sequential text file.

record. When the end of file record is read, it indicates to the program that the update is complete, and the INPUT MASTER is RENAMEd TEST.

The program again asks, "ADD MORE RECORDS Y OR N?", and the user types N for no. The program then CLOSEs OUTPUT MASTER, UNLOCKs OUTPUT MASTER, S6, D2, RENAMEs OUTPUT MASTER, TEST, and LOCKs TEST, S6, D2.

The input and output files are updated. The OUTPUT MASTER, now labeled TEST, is called the CURRENT MASTER, and the cycle begins again at the next update.

You may question, "Why does it seem so slow to have to go through each record during the update?" That is a very valid question. The answer is simply that for a large business the update would be handled by two or more programs. The update information would be read into a program that would sort the update information according to a specific key. Then a program to merge the two files would be run. This is a more efficient method. This book is explaining some of the details of small business programming in BASIC, and does not reach the sophisticated level of applications for large businesses.

Fig. 10-6 is a program to READ (LIST) ALL, OR PARTS OF, A SEQUENTIAL FILE. This program is an example of how a new program is built from sections of previously written programs. The major part of the

```
 MON C,I,O

]RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)?

 ENTER FILE NAME ?TEST

!!! THE NAME IS GOOD
ENTER VALUE FOR SELECTION ELSE

PRESS 'RETURN' TO LIST ALL ENTRIES

ENTER NAME TOKEN ?

ENTER SS NUM TOKEN ?

ENTER DEPT TOKEN ?

ENTER RANGE OF PAY RATE<R>

SELECT ABOVE MINIMUM <M>

SELECT BELOW MAXIMUM<L>

ACCEPT ALL RATES<A>

ENTER SELECTION ?A

PUT IN INPUT SECTION # 1
CATALOG,S6,D1

DISK VOLUME 254

*A 002 DISKINIT
*T 003 TEST

IS CORRECT FILE ON THIS DRIVE Y OR N ?
OPEN TEST,S6,D1
READ TEST
?SECTION INDICATOR
??1
??DATE RECORD
??122181
PR#1
IN#0
```

Fig. 10-7. RUN of the program in Fig. 10-6.

```
DATE OF FILE IS : 122181
PR# 1
READ TEST
?JOHN J. ALDRIGE,000-00-0000,SHIPPING,8
NAME =     JOHN J. ALDRIGE
S SEC # = 000-00-0000
DEPT     = SHIPPING
PAY RATE= 8
----------
?HYMAN T. ALSTON,000-00-0000,SHIPPING,8
NAME =     HYMAN T. ALSTON
S SEC # = 000-00-0000
DEPT     = SHIPPING
PAY RATE= 8
----------
?WILLIAM Z. BASS,000-00-0000,SALES,20
NAME =     WILLIAM Z. BASS
S SEC # = 000-00-0000
DEPT     = SALES
PAY RATE= 20
----------
?HERMAN A. CAMPBELL,000-00-0000,ACCOUNTING,20.5
NAME =     HERMAN A. CAMPBELL
S SEC # = 000-00-0000
DEPT     = ACCOUNTING
PAY RATE= 20.5
----------
?ALBERT O. CUNNINGHAM,000-00-0000,SALES,10
NAME =     ALBERT O. CUNNINGHAM
S SEC # = 000-00-0000
DEPT     = SALES
PAY RATE= 10
----------
?BILL N. FALLS,000-00-0000,ACCOUNTING,10
NAME =     BILL N. FALLS
S SEC # = 000-00-0000
DEPT     = ACCOUNTING
PAY RATE= 10
----------
?JOHN Y. FAULK,000-00-0000,SALES,10
NAME =     JOHN Y. FAULK
S SEC # = 000-00-0000
DEPT     = SALES
PAY RATE= 10
----------
?END SECTION,000-00-0000,00,0
PR#1
IN#0
```

Fig. 10-7—*cont.* RUN of the program in Fig. 10-6.

```
TAKE OUT CURRENT SECTION !

PUT IN INPUT SECTION # 2
CATALOG,S6,D1

DISK VOLUME 254

*A 002 DISKINIT
*T 002 TEST

IS CORRECT FILE ON THIS DRIVE Y OR N ?
OPEN TEST,S6,D1
READ TEST
?SECTION INDICATOR
??2
??DATE RECORD
??122181
PR#1
IN#0

DATE OF FILE IS : 122181
PR# 1
READ TEST
?BILLY F. GARRETT JR.,000-00-0000,SALES,10
NAME =     BILLY F. GARRETT JR.
S SEC # = 000-00-0000
DEPT     = SALES
PAY RATE= 10
----------
?LEUWANDA O. HAMILTON,000-00-0000,VICE-PRESIDENT,50
NAME =     LEUWANDA O. HAMILTON
S SEC # = 000-00-0000
DEPT     = VICE-PRESIDENT
PAY RATE= 50
----------
?LEON B. JENNINGS,000-00-0000,SALES,10
NAME =     LEON B. JENNINGS
S SEC # = 000-00-0000
DEPT     = SALES
PAY RATE= 10
----------
?END RECORD,000-00-0000,00,0
IN#0

]


 PR#0
```

Fig. 10-7—*cont.* RUN of the program in Fig. 10-6.

program in Fig. 10-6 comes from Fig. 8-9 PROGRAM TO READ (LIST) ALL, OR PARTS OF, A SEQUENTIAL TEXT FILE. Fig. 8-9 has no capability to sectionalize a file. The sectionalizing routines were taken from Fig. 10-3 PROGRAM TO UPDATE A SEQUENTIAL FILE, and placed in the program in Fig. 10-6. *The real technique in programming is to synchronize the parts so the program runs*. Fig. 10-7 is a run of the program in Fig. 10-6.

The following lines are taken from Figs. 8-9 and 10-3 to produce the READ PROGRAM, Fig. 10-6.

Fig. 8-9 lines 100–490
Fig. 8-9 lines 570–580
Fig. 8-9 lines 670–740
Fig. 8-9 lines 760–770
Fig. 10-3 lines 12010–12090
Fig. 10-3 lines 12100–12110
Fig. 10-3 lines 21000–21010

The PROGRAM TO READ (LIST) ALL, OR PARTS OF A SEQUENTIAL FILE, Fig. 10-6, performs the same tasks as the program in Fig. 8-9, so the search and list features will not be duplicated. The search and list features, Fig. 8-9, were performed on a different list of employee records than the employee list in Fig. 10-6, but the results would be similar. The figures to be referenced are as follows:

Fig. 8-10A	Search using name token "ZEB".
Fig. 8-10B	Search using the social security token 111.
Fig. 8-10C	Search using a department code token B.
Fig. 8-10D	Search using a range of pay rates.
Fig. 8-10E	Search using above a minimum limit of $5.00 of pay rates.
Fig. 8-10F	Search using below a maximum limit of $9.00 of pay rates.
Fig. 8-10G	Lists all the records in the TEST-RECORDS file.
Fig. 8-11	Using the name token "ZEB" to search a record with the name "Benbow" in the employee name field.

chapter 11

Random Access Files

A random (direct) access file (storage) is a file of fixed record length, and is designed to give relatively fast access time, regardless of the record previously accessed. An employee file with 10,000 records randomly accessed should have a constant access time from one record to the next, no matter where the records are physically located in the file.

DISK STORAGE

Files previous to 1950 were sequentially accessed because the storage medium was magnetic tape. This magnetic tape was coated with mylar, impregnated with ferric oxide particles. About 1950, this same mylar was placed on a circular drum surface. The magnetic disk evolved from the drum. The rotating disk allowed the creation of the random access file. Disks are used as a memory storage device similar to random access memory in the computer (RAM). At the present time, disk storage space is cheaper, and slower, than computer RAM memory. The access time in disks is generally in milliseconds, while RAM memory is in microseconds. The maximum access time for disks is sufficiently short as to be considered direct access storage.

As integrated circuit chip size becomes smaller, and random access memory becomes cheaper, the discussion turns to the idea of computer main memory having over two megabytes storage. This main memory could be

built into the computer, or have an input bus to plug in more random access memory, as it is needed.

FILE TYPES CAN BE SEQUENTIAL OR RANDOM

Random access files can be used for any type of file. Types of files were discussed in Chapter 7, Introduction to Files.

Master, temporary, current master, new master, source, destination, source-destination, and piggyback files can be either randomly or sequentially accessed.

RANDOM ACCESS FILES HAVE A FIXED RECORD LENGTH

Random access files have a fixed record length. Fig. 11-1 shows the comparative makeup of a sequentially accessed, and a randomly accessed record. The example shows a randomly accessed record of 40 characters, and each field has a specific length. The randomly accessed name field is fixed at 16 characters. That 16 character field exactly holds the name, HELEN (space) ZEBRONSKI, and CHR$(13), for the RETURN character. If the name JOHN (space) HO, and the RETURN character, that used eight characters, had been placed in the randomly accessed name field, eight characters of storage space would have been wasted storage space. The randomly accessed file trades wasted space for decreased access time.

The programs, CREATE, UPDATE, and READ a SEQUENTIAL FILE, Figs. 8-2, 8-5, and 8-9, were written as three separate programs to familiarize the reader with the parts necessary to write a complete file program.

RANDOM ACCESS PROGRAM

Fig. 11-2, RANDOM ACCESS PROGRAM, contains the three functions, create, update, and read a randomly accessed file, but it is written as one program. When the RANDOM ACCESS PROGRAM is RUN, the structures within the program allow the user to make the decision to create a new random access file, or use a previously created random access file. The random access file program is a complete program that can be used commercially to handle company files.

NUMBER OF FIELDS AND RECORD LENGTH

When the program is RUN, line 110 causes a jump to the subroutine at line 32000. Line 32000 sets the number of fields equal to nine (NF = 9). The

COMPOSITION OF A RANDOMLY ACCESSED RECORD 40 BYTES LONG

CHARACTER:	J	O	H	N	sp	H	O	cr	0	0	0	-	0	0	0	0	cr	C	6	cr	0	0	.	0	0	cr	0	0	0	0	0	0	0	0	0	0	0
ASCII:	74	79	72	78	32	72	79	13	48	48	48	45	48	48	48	48	13	67	54	13	48	48	46	48	48	13	48	48	48	48	48	48	48	48	48	48
FIELD SIZE:				8										12					3				6							11							

CHARACTER:	H	E	L	E	N	sp	Z	E	B	R	O	N	S	K	I	cr	0	0	0	-	0	0	0	0	cr	D	8	cr	0	0	.	0	0	cr	0	0	0
ASCII:	72	69	76	69	77	32	90	69	66	82	79	78	83	75	73	13	48	48	48	45	48	48	48	48	13	68	56	13	48	48	46	48	48	13	48	48	48
FIELD SIZE:									16											12							3				6					3	

ASCII 13 is a carriage return
Randomly accessed records take more storage space, but the access time from one record to any other record in the file is very fast.

Fig. 11-1. Composition of a randomly accessed record 40 bytes long.

```
100  REM  :PROGRAM TO CREATE A      RANDOM FILE
110  GOSUB 32000
120  GOSUB 20000
130  END
149  REM



150 LK$ =  STR$ (LK)
160  IF  LEN (LK$) < 4 THEN LK$ =  LEFT$ (ZR$,4 -  LEN
     (LK$)) + LK$
170  RETURN
199  REM



200  FOR J = LO TO HI
210 TS$(J) = NP$(J): NEXT J: RETURN
220  REM



240  FOR J = LO TO HI
250  IF  LEN (NP$(J)) < FL(J) THEN NP$(J) = NP$(J) +  LEFT$
     (SP$,FL(J) -  LEN (NP$))
260  NEXT J: RETURN
270  REM



300 SUCC = 1
310  PRINT D$;"READ ";F$;",R";SUCC
320  INPUT LK,CK$: IF SUCC = 1 THEN 350
330  GOSUB 1000: GOSUB 240: IF TS$(SF) =  LEFT$ (NP$(S
     F),TL) THEN DF = 1: RETURN
350 PRED = SUCC:SUCC = LK: IF LK THEN 310
370  RETURN
399  REM



400  HOME : PRINT AC$;" THIS RECORD ?": PRINT : PRINT

440  FOR J = 1 TO NF: PRINT FD$(J);" = ";NP$(J): PRINT

450  NEXT J: PRINT
460  INPUT "ENTER (Y OR N) ?";QU$
470  RETURN
499  REM



500  FOR J = LO TO HI
510  PRINT : PRINT FD$(J);" = ";
520  INPUT QU$
530  IF  LEN (QU$) > FL(J) THEN 510
```

Fig. 11-2. Random access program.

```
540  IF FI THEN 560
550  IF  LEN (QU$) = 0 THEN 570
560 NP$(J) = QU$
570  NEXT J
580 TL =  LEN (NP$(SF)): RETURN
599  REM


600  IF AR = 0 THEN 690
610  PRINT D$;"READ ";F$;",R";AR: INPUT TMP,LK$
620 NODE = AR:AR = TMP:NR = NR + 1: GOTO 695
690 NR = NR + 1:NODE = NR
695  PRINT D$;"WRITE ";F$;",R0": PRINT AR;",";NR: RETURN

699  REM


700 NODE = SUCC
710  PRINT D$;"WRITE ";F$;",R";PRED: GOSUB 150: PRINT
     LK$
720  RETURN
799  REM


800 LK = AR: GOSUB 150: PRINT D$;"WRITE ";F$;",R";NODE
     : PRINT LK$:AR = NODE:NR = NR - 1
810  PRINT D$;"WRITE ";F$;",R0": PRINT AR;",";NR: RETURN

859  REM


950  HOME : VTAB 4: PRINT "ENTER A FIELD TO SEARCH ON
     (1-";NF;") !": PRINT : PRINT
960  INPUT "FIELD =?";SF: IF SF < 1 OR SF > NF THEN 95
     0
970  VTAB 12: PRINT "ENTER A TOKEN TO SEARCH ON !": PRINT
     : PRINT
990 HI = SF:FI = 1:LO = SF: GOSUB 500: RETURN
999  REM


1000 B = 1:E =  LEN (CK$)
1300  FOR J = 1 TO NF - 1
1600 NP$(J) = ""
1900  IF  MID$ (CK$,B,1) =  CHR$ (18) THEN B = B + 1: GOTO
     2500
2200 NP$(J) = NP$(J) +  MID$ (CK$,B,1):B = B + 1: GOTO
     1900
2500  NEXT J:NP$(NF) = "": IF B < E THEN NP$(NF) =  RIGHT$
     (CK$,E - B + 1)
2800  RETURN
2899  REM
```

Fig. 11-2—*cont.* Random access program.

```
4000 PRED = 1:SUCC = PRED
4030  PRINT D$;"READ ";F$;",R";SUCC
4040  INPUT LK,CK$: IF SUCC = 1 THEN 4080
4050  GOSUB 1000: GOSUB 240
4060 RS$ = NP$(1): FOR J = 2 TO NF:RS$ = RS$ + NP$(J):
      NEXT J
4070  IF TR$ < RS$ THEN  RETURN
4080 PRED = SUCC:SUCC = LK: IF LK THEN 4030
4090  RETURN
4099  REM



5000 FI = 0:LO = 1:HI = NF: HOME : PRINT "ENTER NEW VA
     LUE OR PRESS 'RETURN'": PRINT "TO LEAVE VALUE UNC
     HANGED !"
5030 T2 = TL:TMP$ = NP$(SF):TMP = LK:T1 = PRED
5050  GOSUB 500
5060 DF = 0: GOSUB 6000: IF DF = 0 THEN 5200
5090  GOSUB 700
5100  GOSUB 4000
5120 LK = NODE: GOSUB 150: PRINT D$;"WRITE ";F$;",R";P
     RED: PRINT LK$
5140 LK = SUCC: GOSUB 150: PRINT D$;"WRITE ";F$;",R";N
     ODE: PRINT LK$: GOSUB 6500
5160 SUCC = T1: GOTO 5230
5200 LK = TMP: GOSUB 150: PRINT D$;"WRITE ";F$;",R";SU
     CC: PRINT LK$: GOSUB 6500
5230 LK = TMP
5240 TL = T2:NP$(SF) = TMP$: RETURN
5299  REM



6000  GOSUB 200: GOSUB 240:TR$ = NP$(1): FOR J = 2 TO
     NF:TR$ = TR$ + NP$(J): NEXT J
6020  IF PRED = 1 THEN 6065
6030  PRINT D$;"READ ";F$;",R";PRED: INPUT LK$,CK$
6040  GOSUB 1000: GOSUB 240
6050 RS$ = NP$(1): FOR J = 2 TO NF:RS$ = RS$ + NP$(J):
      NEXT J
6060  IF TR$ < RS$ THEN DF = 1: RETURN
6065  IF LK = 0 THEN  RETURN
6070  PRINT D$;"READ ";F$;",R";LK
6080  INPUT LK$,CK$: GOSUB 1000: GOSUB 240
6090 RS$ = NP$(1): FOR J = 2 TO NF:RS$ = RS$ + NP$(J):
      NEXT J
6100  IF TR$ > RS$ THEN DF = 1
6110  RETURN
6199  REM
6500  FOR J = 1 TO NF
6510  FOR K = FL(J) TO 1 STEP  - 1
6520  IF  MID$ (TS$(J),K,1) <  > " " THEN TS$(J) =  LEFT$
     (TS$(J),K): GOTO 6700
6530  NEXT K
6550 TS$(J) = ""
```

Fig. 11-2—*cont.* Random access program.

```
6700  NEXT J
6800  FOR J = 1 TO NF - 1
6850  PRINT TS$(J); CHR$ (18);
6900  NEXT J: PRINT TS$(NF): RETURN
6950  REM
7000  HOME : HTAB 14: PRINT "ALTER ITEMS":AC$ = "ALTER
      "
7030 DF = 0: GOSUB 950:TS$(SF) = NP$(SF): IF TL = 0 THEN
     7230
7060  GOSUB 300
7090  IF DF = 0 THEN 7230
7180  PRINT D$;"PR#0": PRINT D$;"IN#0": GOSUB 400: IF
     QU$ <  > "Y" THEN 7220
7210  GOSUB 5000
7220 DF = 0: GOSUB 350: GOTO 7090
7230  PRINT D$;"PR#0": PRINT D$;"IN#0": PRINT "ALTER S
     EARCH TERMINATED"
7240  INPUT "ANOTHER ALTER SEARCH ? ";QU$
7270  IF QU$ <  > "N" THEN 7030
7300  RETURN
7399  REM


9000  HOME : PRINT  TAB( 10);"PRINT THE FILE": PRINT "
     STOP AFTER EVERY RECORD (Y OR N) ?";: INPUT " ";Q
     U$: IF QU$ <  > "N" AND QU$ <  > "Y" THEN 9000
9010  PRINT D$;"READ ";F$;",R1": INPUT PRED: IF PRED =
     0 THEN  PRINT D$;"PR#0": PRINT D$;"IN#0": GOTO 92
     80
9020  PRINT D$;"READ ";F$;",R";PRED
9040  INPUT SUCC,CK$: PRINT D$;"PR#0": PRINT D$;"IN#0"

9100  GOSUB 1000
9180  FOR J = 1 TO NF
9200  PRINT FD$(J);" = ";NP$(J)
9220  NEXT J
9230  IF QU$ = "N" THEN 9260
9240  PRINT : INPUT "ENTER Q TO QUIT ELSE PRESS RETURN
      !";Q$: IF Q$ = "Q" THEN 9280
9260  IF SUCC > 0 THEN PRED = SUCC: GOTO 9020
9280  RETURN
9299  REM


10000  HOME : HTAB 15: PRINT "ADD RECORDS"
10100 FI = 1:LO = 1:HI = NF:SF = 1
10300  GOSUB 500: IF TL = 0 THEN 12100
10310  GOSUB 200: GOSUB 240
10350 TR$ = NP$(1): FOR J = 2 TO NF:TR$ = TR$ + NP$(J)
     : NEXT J
10400  GOSUB 600: GOSUB 4000
10500 LK$ = NODE: GOSUB 150: PRINT D$;"WRITE ";F$;",R";
     PRED: PRINT LK$
10600  PRINT D$;"WRITE ";F$;",R";NODE
```

Fig. 11-2—*cont.* Random access program.

```
10900 LK = SUCC: GOSUB 150
11200  PRINT LK$
11500  FOR J = 1 TO NF - 1: PRINT TS$(J); CHR$ (18);: NEXT
     J: PRINT TS$(NF)
11800  PRINT D$;"PR#0": PRINT D$;"IN#0"
12100  HOME : VTAB 3: INPUT "DISCONTINUE ADDING ITEMS
     ?";QU$
12400  IF QU$ <  > "Y" THEN  GOTO 10300
12700  RETURN
12999  REM



13000  HOME : HTAB 14: PRINT "DELETE ITEMS":AC$ = "DEL
     ETE"
13300 DF = 0: GOSUB 950: IF TL = 0 THEN 16600
13900 TS$(SF) = NP$(SF): GOSUB 300
14200  PRINT D$;"PR#0": PRINT D$;"IN#0": IF DF = 0 THEN
     16000
15100  GOSUB 400
15400  IF QU$ <  > "Y" THEN 15800
15700  GOSUB 700:TMP = LK: GOSUB 800:LK = TMP: IF LK =
     0 THEN 15900
15800 DF = 0:SUCC = PRED: GOSUB 350: GOTO 14200
15900  PRINT D$;"PR#0": PRINT D$;"IN#0"
16000  PRINT "DELETION ON SEARCH TERMINATED": PRINT : PRINT

16100  INPUT "TRY ANOTHER DELETION ?";QU$
16300  IF QU$ <  > "N" THEN 13300
16600  RETURN
16899  REM



16900  HOME : HTAB 10: PRINT "FIELD CLARIFICATION"
17200  PRINT : PRINT "FIELD #"; TAB( 10);"FIELD NAME";
      TAB( 35);"LENGTH": PRINT : PRINT
17500  FOR J = 1 TO NF
17800  PRINT J; TAB( 10);FD$(J); TAB( 35);FL(J)
18100  NEXT J
18400  PRINT : PRINT : PRINT "TOTAL RECORD LENGTH IS "
     ;RL
18700  PRINT : PRINT "PRESS ANY KEY TO CONTINUE ...";:
     GET A$
19000  RETURN
19999  REM



20000  GOSUB 31000
20300  HOME : PRINT "DO YOU WANT TO BUILD A NEW FILE":
      PRINT : PRINT "OR USE AN EXISTING FILE???"
20600  PRINT : PRINT : INPUT "ENTER N(NEW) OR O(OLD)?
     ";QU$
20900  IF QU$ = "N" THEN  GOSUB 31170: GOTO 21800
21200  IF QU$ = "O" THEN  GOSUB 29700
```

Fig. 11-2—*cont.* Random access program.

```
21500  IF QU$ <  > "O" AND QU$ <  > "N" THEN 20300
21800  HOME : PRINT "SELECT FUNCTION FROM FOLLOWING LI
     ST": PRINT : PRINT : PRINT
22100  VTAB 8: PRINT "1. ADD ITEMS"
22400  PRINT : PRINT "2. DELETE ITEMS"
22700  PRINT : PRINT "3. ALTER ITEMS"
23000  PRINT : PRINT "4. PRINT THE LIST"
23300  PRINT : PRINT "5. EXAMINE THE ";NF;" FIELDS"
23600  PRINT : PRINT "6. CLOSE THE FILE AND END"
23900  PRINT : PRINT : INPUT "ENTER REQUEST ? ";SE
24200  IF SE < 1 OR SE > 6 THEN 21800
24500  ON SE GOSUB 10000,13000,7000,9000,16900
24800  IF SE <  > 6 THEN  GOTO 21800
24900  REM



25100  PRINT D$;"WRITE ";F$;",RO"
25400  PRINT AR;",";NR
25700  PRINT D$;"CLOSE"
25800  PRINT D$;"LOCK ";F$;",S";SL;",D";DR
26000  RETURN
26999  REM



29700  PRINT D$;"UNLOCK ";F$;",S";SL;",D";DR
30000  PRINT D$;"OPEN ";F$;",L";RL;",S";SL;",D";DR
30010  PRINT D$;"READ ";F$;",RO"
30020  INPUT AR,NR
30030  PRINT D$;"PR#O": PRINT D$;"IN#O": RETURN
30999  REM



31000 D$ =  CHR$ (4)
31010 SL = 6: REM :SLOT IS #6
31020 DR = 1: REM :DRIVE IS #1
31030  HOME : VTAB 3
31040  PRINT "UNIT ACCESSED IS SLOT ";SL;" DRIVE ";DR
31050  PRINT
31060  INPUT "CHANGE THESE VALUES(Y OR N)? ";QU$
31070  PRINT
31080  IF QU$ <  > "Y" THEN 31130
31090  INPUT "ENTER SLOT (1-7) ?";SL
31100  IF SL < 1 OR SL > 7 THEN 31090
31110  INPUT "ENTER DRIVE (1-2) ?";DR
31120  IF DR <  > 1 AND DR <  > 2 THEN 31110
31130  PRINT : INPUT " ENTER FILE NAME ?";F$
31140  IF  LEN (F$) = O THEN  PRINT : PRINT "*** NAME
     IS INVALID ***": GOTO 31130
31150  FOR J = 1 TO  LEN (F$): IF  ASC ( MID$ (F$,J,1)
     ) < 32 THEN  PRINT : PRINT "*** PLEASE DON'T TRY
     CONTROL CHARA-": PRINT "CTERS YET !": GOTO 31130
31160  NEXT J: PRINT : PRINT "!!!THE NAME IS GOOD": RETURN

31169  REM
```

Fig. 11-2—*cont.* Random access program.

```
31170  PRINT D$;"OPEN ";F$;",L";RL;",S";SL;",D";DR
31180  PRINT D$;"WRITE ";F$;",R0"
31190  PRINT "0,1"
31210  PRINT D$;"WRITE ";F$;",R1"
31220  PRINT "0000,": PRINT  LEFT$ (SP$,FL(1))
31240 AR = 0:NR = 1
31250  PRINT D$;"IN#0": PRINT D$;"PR#0"
31260  RETURN
31999  REM


32000 NF = 9: REM - NUMBER OF FIELDS
32010  DIM FL(NF),FD$(NF),NP$(NF),TS$(NF)
32070  FOR J = 1 TO NF: READ FD$(J),FL(J): NEXT J
32080 SP$ = "            ":SP$ = SP$ + SP$ + SP$ + SP$:Z
     R$ = "000"
32090 RL = 5 + NF: FOR J = 1 TO NF:RL = RL + FL(J): NEXT
     J: RETURN
32140  DATA  LASTNAME,12
32150  DATA  FIRSTNAME,8
32160  DATA  STREET,15
32170  DATA  CITY,10
32180  DATA  STATE,2
32190  DATA  ZIP CODE,5
32200  DATA  PHONE NUMBER,12
32210  DATA  PURCHASE DATE,8
32220  DATA  MEMO,44
32230  REM :THESE 9 DATA STATEMENTS SPECIFY THE DATA F
     IELDS
```

Fig. 11-2—*cont.* Random access program.

variable dictionary for the program may be viewed in Chart 11-1. These fields are placed in DATA statements and READ using a loop (FOR J = 1 TO NF) and a READ statement. The nine fields are as follows.

1. Lastname, contains up to 12 characters
2. Firstname, contains up to 8 characters
3. Street, contains up to 15 characters
4. City, contains up to 10 characters
5. State, contains up to 2 characters
6. Zip code, contains up to 5 characters
7. Phone number, contains up to 12 characters
8. Purchase date, contains up to 8 characters
9. Memo, contains up to 44 characters of information

Chart 11-1. Variable Dictionary for the Random Access Program (Alphabetical Order)

AC$	Action string—holds the label telling what action is being taken. AC$ has two possible values, "DELETE", or "ALTER".
AR	Available record pointer—AR is equal to zero, unless a record, or records have been deleted. In case of deletion, AR points to the space where the record was deleted.
B	Is a counter. It begins counting at the first character of CK$, and is used to examine each character in CK$.
CK$	Is used to read the record in packed form. Packed form means having all trailing blanks removed. This is done to save time on the transfer rate when reading the record from disk.
D$ = CHR$(4)	D$ is used to alert DOS to the command.
DF	Delete flag. When DF = 0, it means no record was found that matched the search token. When DF = 1, it means a record has been found that matches the search token.
DR	Is the drive number to which the file is written, or read.
E	Is the variable that holds the number of characters in CK$.
F$	Holds the name of the file (name) being manipulated.
FD$	Field descriptor. The FD$ describes each of the fields, LASTNAME, FIRSTNAME, ADDRESS, etc.
FI	Flag Input. The Flag Input is used by the subroutine at lines 500–580. FI equals 0 means the record is being altered, otherwise, it means the record is being entered for the first time.
FL	Field Length. FL holds the maximum number of characters of each field contained in NP$. NP$(1) holds the last name (FD$(1)—field descriptor, and FL(1) = 12).
HI	Is a variable to indicate the high range of the fields to be operated on.
J	Loop variable.
K	Loop variable.
LEN(LK$)	Always contains four (4) digits. See LK$.
LK$	LK$ contains a four digit numeric string variable having a value of LK. LK is a link pointer. Line 150—LK$ = STR$(LK). The value held in LK points to the next record in the file. LK$ is four places to keep the record layout consistent. In the random access record, LK (4 digits) followed by a return character (CHR$(13)) makes up a total of five characters.
LO	Is a variable to indicate the low range of fields to be operated on.
NF	Is the number of fields in the record. Line 32000 NF = 9. In the Random Access Program, there are nine fields in the record.
NODE	NODE IS A POINTER, this is the record number of the current record being added to the file.
NP$	Storage location to hold a record. A new record to be added to the file is input into NP$. A record on disk is placed in NP$ before it is placed on the screen.
NR	Is the number of records in the file.
PRED	PRED IS A POINTER (Predecessor). The record preceding the record being examined.

Chart 11-1—*cont.* Varible Dictionary for the Random Access Program (Alphabetical Order)

PRINT D$;"IN#0"	Tells the disk operating system (DOS) to get all input from the keyboard.
PRINT D$"PR#0"	Tells the disk operating system (DOS) to put all PRINT statements on the CRT.
Q$	Query to interactive question after listing a single record in OPTION No. 4.
QU$	Question. Holds the response to the interactive question.
R	R is not a variable. R is the record number being read from, or written to, the disk.
RL	Record length. RL is the sum of the lengths of all the records plus five characters, plus the number of fields. Five is the four position pointer (LK$), plus the one return character (CHR$(13)), following LK$.
RS$	Record storage string. RS$ is used to store a record in expanded form. Expanded form means all fields are at their full length, and stored in a single variable, RS$. This is in contrast to the packed form used when the record is stored on disk (CK$).
SE	Holds the value of the selection entered from the main menu.
SF	Search field. SF is used to indicate which field is being checked for a match for the token, in the search routine.
SL	Slot. The slot number in which the disk is to receive the random access file.
SP$	Is a string array containing 40 spaces. SP$ is used to fill the unused spaces in NP$. When searching for a specific field in NP$, the field will be filled with the correct number of characters and spaces.
STR$(LK)	STR$ converts the value held in LK into a string. The LK$ is always followed by a return character (CHR$(13)). If LK$ was not a constant length, DOS would not be able to reconstruct the record correctly.
SUCC	SUCC IS A POINTER (Successor pointer). The successor is the record following the predecessor record.
T1	Temporary storage variable.
T2	Temporary storage variable.
TL	Token length. TL is the length of the token being searched on. If the token is "Bob", TL equals 3.
TMP	Temporary location. TMP is used for various things such as pointer values that will be changed when deleting a record, or pointer values that do not need to be stored when the records data portion is needed. TMP contains a real value, and is different from the temporary location TS$.
TMP$	Temporary string storage.
TR$	Temporary record string. TR$ is the record in expanded form to be added to the file. A comparison of TR$ and RS$ is an accurate method to determine where the new record should go in the file. The record can be placed at the end of the file, unless there has been a deletion.
TS$	Temporary storage variable to hold values for each of the fields until they can be placed on disk. TS$ is different from TMP.
ZR$	Is a four place string field filled with 4 zeros.

RECORD LENGTH

The total record length is 130 characters in length, Chart 11-2. The record is composed of the following:

1. ZR$ is composed of four digits (0000) and is used as a pointer that points to a specific record in the file—4 characters.
2. A return character follows ZR$, and is the fifth character in the record (CHR$(13))—1 character.
3. The nine fields make a total of 116 characters.
4. Each field is followed by a delimiter, for a total of nine characters.

Chart 11-2. Random Access Record (130 Characters Long)

POINTER = 0000,		
ZR$ = "0000" + RETURN	=	5 CHARACTERS
ITEM—CONTAINS NINE FIELDS		
1. LASTNAME	=	12 CHARACTERS
2. FIRSTNAME	=	8 CHARACTERS
3. STREET	=	15 CHARACTERS
4. CITY	=	10 CHARACTERS
5. STATE	=	2 CHARACTERS
6. ZIP CODE	=	5 CHARACTERS
7. PHONE NUMBER	=	12 CHARACTERS
8. PURCHASE DATE	=	8 CHARACTERS
9. MEMO	=	44 CHARACTERS
ONE RETURN CHARACTER AFTER EACH FIELD	=	9 CHARACTERS
TOTAL NO. OF CHARACTERS		130 CHARACTERS

FLEXIBILITY OF THE RANDOM ACCESS PROGRAM

The Random Access Program was written so the number of fields and the length of each field can be easily changed.

The number of fields can be changed by changing line 32000. If 12 fields are needed in the program, line 32000 should be changed to NF = 12. The field descriptor name and length of the field descriptor must be added in the data statements. For example, if a customer's bank, bank account number, and personal reference at the bank were added to the program, it could be done in the following manner.

```
32000 NF = 12 : REM-NUMBER OF FIELDS
   •
   •
   •
```

```
•
•
32230 DATA CUSTOMER'S BANK,15
32240 DATA BANK ACCOUNT NUMBER,11
32250 DATA PERSONAL REFERENCE, 22
32260 REM-THESE 12 DATA STATEMENTS SPECIFY THE DATA FIELDS
```

The field descriptor name and length of the field descriptor in lines 32140–32220 can be changed in any way to accommodate the needs of a specific business.

APPLE STRING LIMITED TO 256 CHARACTERS

The one limitation is that the record must be less than 256 characters. After the record is manipulated, it is stored in expanded form in a single string, RS$. Expanded form is the form used when the record contains 130 blanks and characters. Expanded form is contrasted to packed form, where no trailing blanks are stored in the record. The string variable for the packed form is CK$. The Apple computer allows a maximum of 256 characters in a string.

DATA BASE

The Random Access Program is used as an introduction to the DATA BASE concept. A DATA BASE is a file of data so structured that each department of a company can access one data file. A single data file, accessed by all users, ensures greater accuracy.

Not all departments use the same information. Management needs sales, production, and profit results. Accounting needs accounts receivable, accounts payable, payroll, and credit information. Inventory needs parts in stock, parts on order, and parts on back order.

With all information in a single data base, each department can have a security clearance to access specific fields of the data base, while the other fields cannot be accessed. The important aspect is that all users have one single source of information. This single source of information is easier to check for accuracy and completeness, than if each department had its own data base.

The data base concept is an ideal method to store data, but a single data base accessible to all departments is difficult to implement in the business world.

The program returns to line 120, which is GOSUB 20000. Line 20000 is GOSUB 31000.

LINKED LIST

The Random Access Program uses a linked list structure to maintain the proper order of the records in the file.

1. The record consists of the pointer and the item.
2. Pointer—four digit value at the beginning of each record.
    a. NODE is a pointer that points to the present record.
    b. PRED is a pointer that points to the preceding record.
    c. SUCC is a pointer that points to the succeeding record.
3. Item—the data held in the nine fields of the record.

A linked list is defined as a flexible scheme in which each record stores a pointer value of the next record (the next record number). Each record in the file is called a node. The variable, NODE, used in the Random Access Program is used to store the record value for certain operations, such as adding a new record, and deleting a record.

In the Random Access Program, each record is placed in the file based on the contents of the fields. For example, SMITH, JOHN (one item) is placed before SMITHE, JOHN. SMITH, JOHN is placed after SMITH, CHARLES.

In order to allow the user to enter the records in any order, and have the program write the records in the correct order, a linked list is maintained.

When the new record, SMITH, JOHN, is entered, it is assembled in the variable, TR$, so that SMITH is followed by seven spaces (FL(1) = 12), then JOHN followed by four spaces (FL(2) = 8), etc. The other seven fields in the item follow the same pattern. This is done so the comparison of the new record and the records on file are consistent.

FUNCTION OF SUBROUTINE AT 31000—SLOT AND DRIVE OPTIONS

The subroutine at lines 31000 through 31169 set up the disk operating system command, D$ = CHR$(4). It also gives the user the option of using a fixed slot #6 and drive #1 to place the disk that is to receive the random access file data, or allows the user to select the slot and drive of his or her choice.

FILE NAME—NEW FILE OR OLD FILE

When the desired slot and drive have been selected to place the disk on which the random access file is to be written to, or read from, the program

asks for the file name to be entered. The name entered in line 31130 may be used to build a new file (option in line 20300), or to open a file created previously.

The file name is checked in two ways. If no file name is entered, and RETURN is pressed (31140 IF LEN(F$) = 0 THEN PRINT : PRINT "!!!NAME IS INVALID" : GOTO 31130), the program branches to line 31130 to ask for another name. Line 31150 checks to determine if the file name contains control characters. If the file name contains control characters, the file name is invalid, and the program jumps to line 31130 to ask for another name. If a legal file name is entered, the program prints out, "!!!THE NAME IS GOOD", and the program returns to line 20300.

BUILD A NEW FILE OR USE AN EXISTING FILE?

Line 20300 asks the user, "DO YOU WANT TO BUILD A NEW FILE OR USE AN EXISTING FILE???". The interactive question, line 20600, allows the user the option to enter his or her selection, "N(EW), O(LD)".

CREATE A NEW FILE

If the user selects the N(EW) option, the program calls the subroutine at line 31170 (GOSUB 31170). The subroutine at lines 31170 through 31999 sets up the beginning record information, the "DUMMY" header record, and assigns the available record pointer (AR = 0) a value of zero (0), and the number of records in the file (NR = 1, this is the "DUMMY" record) a value of one (1).

Line 31170 opens the file using the name entered in line 31130. Line 31190 PRINT "0,1", prints the values "0,1", to the disk to indicate that the file contains one record, and the value of the available record pointer is zero.

```
31220 PRINT "0000," : PRINT LEFT$(SP$,FL(1))
```

The PRINT "0000," sets the four digits to be used in the pointer. Four digits are specified so the record pointer will be consistent to a value of 9999 records. If the LASTNAME field contains, SMITH, and the pointer variable (LK) changes from 99 to 100, the first letter of SMITH is written over. When the pointer variable (LK) is changed from 100 to 1000, the second letter in SMITH is written over.

```
9  9  13  S  M  I  T  H
1  0  0  13  M  I  T  H
1  0  0  0  13  I  T  H
```

Conversely, when the pointer changes from four digits to three digits, DOS is confused, and the entire item is dropped from the record.

```
1  0  0  0 13  S  M  I  T  H
1  0  0 13 13  S  M  I  T  H
```

(Nothing would be put in CK$, and would cause the program to end abnormally.)

PRINT "0000" EQUALS FIVE CHARACTERS

The "0000" actually contains five characters. There is no semicolon following "0000", so the line is closed out. When a line is closed out, a return character (CHR$(13)) is placed at the end of the line. In line 32090, RL = 5 + NF. The total record length equals five (0000,), plus the number of fields.

DUMMY HEADER RECORD

The second statement in line 31220 sets the "DUMMY" header record equal to the number of spaces in field length one (PRINT LEFT$(SP$,FL(1)). Line 31240 (AR = 0 : NR = 1) assigns the value of zero to the available record pointer, and the value of one (1) to the number of records in the file. Line 31250 tells DOS to get all input from the keyboard (PRINT D$;"IN#0"), and to send all PRINT statements to the CRT (PRINT D$;"PR#0"). Line 31260 RETURNS to the last statement in line 20900, which is GOTO 21800. GOTO 21800 causes the program to jump to the main menu of the program. The main menu contains six options for file maintenance.

FUNCTION OF SUBROUTINE AT 29700 IF THE OLD FILE OPTION IS SELECTED

If the user wishes to access a file previously opened, "O" for OLD is typed in answer to the interactive question in line 21200. This choice causes GOSUB 29700 to be executed. The subroutine at lines 29700 through 30999 sets all the information necessary to do file maintenance on a previously opened file (file name was entered in line 31130), and the slot and drive options were selected in lines 31000 through 31120. All files are CLOSEd and LOCKed at the termination of the program, so all files must be UNLOCKed, line 29700, before they can be used.

Line 30000 opens the file, and tells DOS that the file is a random access file by giving the record length—RL is defined in line 32090.

```
30000 PRINT D$;"OPEN ";F$;",L";RL;",S";SL;",D";DR
```

SEQUENTIAL FILE STATEMENT (PRINT D$;"OPEN";F$;",S";SL;",D";DR)

When a random access file is used, DOS must know the length of the record so it can search the file to read and write the correct record, on the disk.

```
30010 PRINT D$;"READ ";F$;"R0"
```

Record zero (R0) holds the values of the available record pointer (AR), and the number of records in the file (NR).

The values stored in record zero (R0) are assigned to the available record pointer in the file at line 30020.

Line 30020 (INPUT AR,NR) reads the value stored in the available record pointer, and the number of records from the disk. Line 30030 tells DOS to get all input from the keyboard and send all PRINT statements to the CRT. The subroutine returns to line 21500 which is FALSE (the file opened was "OLD"), so the program defaults to line 21800. HOME clears the screen, and the six options in the main menu are placed on the screen.

MAIN MENU HAS SIX OPTIONS

The main menu has six options.

1. ADD ITEMS
2. DELETE ITEMS
3. ALTER ITEMS
4. PRINT THE LIST
5. EXAMINE THE ;NF; FIELDS
6. CLOSE THE FILE AND END

Line 23900 allows the user to enter one of the six selections for file maintenance. Line 24200 is a check to determine if a legal value of 1 to 6 was selected. If an illegal value, less than 1, or greater than 6,. is input, the program branches to line 21800 to display the menu, and ask the user for another selection.

```
24500 ON SE GOSUB 10000, 13000,7000,9000,16900
```

Line 24500 causes the program to branch to the subroutine selected by the user. There is a direct correlation between the number selected and the

line number of the subroutine. Selection of 1 causes a branch to the subroutine beginning at line 10000, selection 2 causes a branch to the subroutine that begins at line 13000, selection 3 causes a branch to the subroutine that begins at line 7000, selection 4 causes a branch to the subroutine that begins at line 9000, and selection 5 causes a branch to the subroutine that begins at line 16900. There is no specific routine for selection number 6.

```
24800 IF SE <> 6 THEN GOTO 21800
```

Each subroutine returns to line 24800. If the selection value is from 1 to 5, the statement is TRUE, and the program branches to line 21800 to display the menu.

If the selection value is 6, line 24800 is FALSE, and the program defaults through lines 24500, and 24800, to the routine that begins at line 25100, to print the available record pointer value, and number of records in the file, to disk, CLOSEs and LOCKs the file, and returns to line 130, to END the program.

OPTION NO. 1, ADD ITEMS

When the user selects option number 1 from the menu, ADD ITEMS, the program branches to the subroutine that begins at line 10000. Line 10000 clears the screen (HOME), and adds the prompt, ADD RECORDS, on the screen.

Function of Subroutine at 10000

```
10100 FI = 1 : LO = 1 : HI = NF : SF = 1
10300 GOSUB 500 : IF TL = 0 THEN 12100
```

FI = 1 means the flag input value is set to 1. When the flag input is set to 1, it indicates the record is being entered. FI = 0 means the record is being altered.

LO = 1 means that one is the value stored in the start value of the loop range.

HI = NF—NF was assigned a value of 9 in line 32000. The values held in the variables LO and HI are used as the beginning and ending limits in the loop in the subroutine that begins at line 500.

SF = 1—when the search flag variable is assigned a value of 1, line 580 assigns the length of field one to TL.

Function of Subroutine at 500

The subroutine that begins at line 500 allows the user to input the nine data items that form part of the random access record and items. The record is

the pointer that consists of five characters (0000,), and the nine data items in the field descriptors, and the nine delimiters that follow each of the nine fields.

Line 500 (FOR J = LO TO HI) is the beginning of a loop that increments to set up the field descriptors to allow the user to input the data items into the record.

Line 510 causes a line to be skipped (PRINT), and prints the appropriate field descriptor on the screen.

```
FD$(1) LASTNAME = ???????????? (ENTER LASTNAME)
```

Line 520 (INPUT QU$) allows the user to input the specific data field. For example, FD$(1)—LASTNAME = asks the user to input the last name to be placed in the file. Nine data fields are entered in the ITEM.

```
530 IF LEN(QU$) > FL(J) THEN 510
```

If the length of the field entered by the user is greater than the field length (FL(1) = 12) the program branches to line 510 to allow the user to enter a legal name that contains the proper number of characters.

```
540 IF FI THEN 560
```

This is a logic statement, equivalent to IF FI > 0, the statement is TRUE, or IF FI = 0, the statement is FALSE. When adding a record for the first time, FI = 1. If FI = 1, and there is nothing placed in the field, RETURN is pressed and the field is accepted as a null value.

When altering a record FI = 0, this will change a field only when data is entered into QU$.

```
560 NP$(J) = QU$
```

When line 540 is TRUE (FI = 1), the program branches to line 560 to assign the item entered into QU$, to the correct field of the record string (NP$).

Line 580 stores the token length entered in the search field (SF). Line 580 is used when entering a token to search for a record in the file. When adding a record to the file, SF = 1, giving the token length of the last name entered. The SF variable contains a value from 1 to NF, so a specific field can be searched on.

The RETURN in line 580 causes the program to continue with the second statement in line 10300, which is, IF TL = 0 THEN 12100. This ensures that at least one character was entered in field 1. If no character was entered

in field 1, then the ADD routine is bypassed allowing the user to discontinue the ADD function.

```
10310 GOSUB 200 : GOSUB 240
```

Function of Subroutine At 200

The subroutine that begins at line 200 transfers the storage location variable (NP$) to the temporary storage variable (TS$), in order to save the field values in their original form, as they were input.

Function of Subroutine At 240

The subroutine that begins at line 240 space fills all the fields in the record. If the length of the field is less than its maximum length, spaces are added to fill the field.

```
10350 TR$ = NP$(1) : FOR J = 2 TO NF : TR$ = TR$ + NP$(J) : NEXT J
```

Line 10350 takes all the fields in the item, and concatenates them into a single string variable (TR$).

```
10400 GOSUB 600 : GOSUB 4000
```

Function of Subroutine At 600

The subroutine that begins at line 600 finds a free record pointer (AR), and assigns that free record to a NODE. There are two cases handled by the subroutine at lines 600 through 699. (1) There are no record vacancies in the file. (2) There are record vacancies between records.

```
600 IF AR = 0 THEN 690
```

Case No. 1—line 600 determines if case No. 1, or case No. 2 applies. If the available record pointer (AR = 0) equals zero, the program knows there are no record vacancies within the file, and branches to line 690, to increment the number of records, assign the number of records in the file to the NODE variable. The NODE is always assigned the value where the next record is to be placed.

Line 695 writes record number zero (R0 is where AR and NR are stored) to the file to update the available record pointer, and the number of records in the file. The program RETURNs to continue processing.

Case No. 2—a deletion has been made from the file leaving a record vacancy in the file. Line 610 reads the record at the available record pointer, inputs from disk the link to the next available (TMP), and inputs the old item value (LK$) but discards it so the system will not hang.

Line 620 NODE is set to the available record pointer, the available record pointer is set to TMP, the record count is incremented, and the program jumps to line 695. Line 695 writes the value of AR and NR into R0, and RETURNs to continue processing.

Since this is the ADD ITEM routine, the program returns to the second statement in line 10400, which is GOSUB 4000.

Function of Subroutine At 4000

The subroutine at 4000 reads through the file, and determines where the new record should be linked in the file. The preceding record (PRED), and the succeeding (SUCC), are set to 1 which contains the "DUMMY" header record. Line 4030 reads the successor (SUCC) record, the link (LK) to the next record, and the item value (CK$) are input. If SUCC = 1 (you are reading the "DUMMY" header record), the program branches to line 4080. Line 4080 assigns the value of the successor record number into the predecessor, and the link value into the successor record number. If the link (LK) is not zero, the program branches to line 4030. When link (LK) is zero, you have reached the end of the file, and the record to be added will be linked at the end of the file.

In line 4040 IF SUCC is not 1, the program defaults to the subroutine at line 1000.

Function of Subroutine At 1000

The subroutine at 1000 takes the item value in CK$, and breaks it apart putting each field in NP$.

```
1000 B = 1 : E = LEN (CK$)
```

B (BEGINNING) is set to the first character position in CK$. E (END) is set to the total numbers in CK$.

```
1300 FOR J = 1 TO NF - 1
```

The "J" loop reconstructs the values of all fields except the last one. The delimiter following the first eight record fields is CHR$(18). Any value after the eighth CHR$(18), and before the RETURN character, is the value of the last field. (CHR$(18) is written in line 11500.)

```
1600 NP$(J) = ""
```

Line 1600 sets the field to a null value. NP$(J) ="" is to clear the field of any value from a previous usage.

Line 1900 checks CK$ to look for the CHR$(18) delimiter. If the CHR$(18) delimiter is discovered, at character position "B" (IF is true),

then B is incremented, and the program branches to line 2500 and causes the loop to set up for the next field in the record.

If line 1900 is FALSE, then it defaults to line 2200 which concatenates the character at position "B" onto NP$(J). It then increments "B", and jumps to line 1900 to check the next character. This loop recreates NP$(1) through NP$(8).

The second statement in line 2500 (NP$(NF) = "") clears the ninth field in the record.

The last statement in line 2500, IF B < E THEN NP$(NF) = RIGHT$(CK$,E − B + 1), sets the ninth field to all the characters from B to E. For example, if E = 100, and B = 90, transfer the rightmost 11 characters (E − B + 1) into NP$(NF). The program returns to the second statement in line 4050, which is GOSUB 240, to pack the fields with spaces to their full length.

Return to Subroutine 4000 (Line 4060)

At line 4060, the expanded form of the record is stored in a single variable (RS$).

Line 4070 compares the record stored in RS$ (Line 4060) to the new record in temporary storage (TR$). If the record to be added is less than the record on disk, alphabetically or numerically (compares string characters), the program returns to line 10500.

If the record is greater than or equal to (TR$ >= RS$), the SUCC value is assigned to PRED, the link (LK) value is assigned to the successor value (SUCC). If the link is greater than zero, the program branches to line 4030 to get the next record, in order to compare the item value on the disk to the item value of the record to be added.

```
10500 LK = NODE : GOSUB 150 : PRINT D$;"WRITE ";F$;",R";PRED : PRINT LK$
```

LK = NODE—the record number of the record being added is assigned to the link pointer (LK). The program then jumps to the subroutine that begins at line 150.

Function of Subroutine At 150

```
150 LK$ = STR$(LK)
```

Line 150 converts the numeric value stored in the link pointer (LK) to a string representation. The link pointer (LK) is converted to a string to make sure it always contains four digits.

```
160 IF LEN(LK$) < 4 THEN LK$ = LEFT$(ZR$,4 − LEN(LK$)) + LK$
```

Line 160 adds as many zeros as necessary to ensure that LK$ contains four digits. This is done to ensure that the first byte of the ITEM starts at the sixth byte of the record. If the pointer, and the RETURN character, do not occupy the first five bytes of the record, the record will not be read from disk correctly. The program then RETURNs to the third statement in line 10500 (PRINT D$;"WRITE ";F$;",R";PRED).

The third statement in line 10500 changes the link pointer value in the predecessor (PRED) record to point to the record being added (NODE). PRINT LK$ prints the four digits of the pointer, and a RETURN character, to the disk.

```
10600 PRINT D$;"WRITE ";F$;",R";NODE
```

Line 10600 tells DOS to write the link pointer (LK), and ITEM into the record NODE. Line 10900 converts the succeeding (SUCC) record pointer into a four position string. LK is first assigned the value stored in SUCC, and then the subroutine that begins at line 150 assigns the numeric value stored in the link pointer (LK) into a string representation (LK$).

```
11200 PRINT LK$
```

Line 11200 prints the pointer value (LK$) into the first five positions of the record NODE.

Line 11500 writes the first eight fields to the record stored on disk, and each field is followed by a CHR$(18). The ninth field is also written to disk. This concludes the process of adding a record (pointer and item) to the file.

```
11800 PRINT D$;"PR#0" : PRINT D$;"IN#0"
```

Line 10800 tells DOS that all PRINT statements (PR#0) go to the CRT, and all input (IN#0) is to come from the keyboard.

Lines 12100 and 12400 related to the interactive question, DISCONTINUE ADDING ITEMS?

The program then RETURNS to line 24800, IF SE <> 6 THEN GOTO 21800. Since option No. 1, ADD ITEMS, was selected, the menu is printed on the screen.

OPTION NO. 2, DELETE ITEMS

Selection of option No. 2, DELETE ITEMS, from the menu, causes the program (line 24500) to branch to the subroutine that begins at line 13000.

Function of Subroutine At 13000

Line 13000 clears the screen, and prints, DELETE ITEMS. The action string (AC$) is assigned the literal "DELETE".

```
13300 DF = 0 : GOSUB 950 : IF TL = 0 THEN 16600
```

If the search routine finds a record that has a field which matches the search token, then the delete flag is set to 1 (DF = 1). Before the search is begun, the delete flag must be set to 0 (DF = 0), so that the value of DF, upon returning from the search routine, will be correct.

Function of Subroutine At 950

```
950 HOME : VTAB 4 : PRINT "ENTER A FIELD TO SEARCH ON (1 - ";NF;")!" : PRINT
: PRINT

960 INPUT "FIELD =?";SF : IF SF < 1 OR SF > NF THEN 950
```

Line 960 asks the user to enter the field number (1–9) on which the search is to be conducted. A check is made to determine if the field value is in the legal range (1–9).

Line 970 asks the user to enter the token on which to search.

```
990 HI = SF : FI = 1 : LO = SF : GOSUB 500 : RETURN
```

Line 990 assigns the value to the search field variable, to be used for the high index of the loop. The flag input variable is set to 1 (FI = 1), to allow entry of a token of 0 length. A token of 0 length indicates the user does not want to continue the search, in case a mistake was made in selecting the DELETE ITEMS from the menu.

LO = SF—the value in the search field is assigned to the low index of the loop. This means that the loop increments only one time on the selected field. The program jumps to the subroutine at line 500. The subroutine at line 500 allows the actual input of the token—see function of subroutine at line 500. The program RETURNs to the third statement in line 13300, which is, IF TL = 0 THEN 16600.

If the length of the token entered is equal to 0 (TL = 0), the program jumps to line 16600, which is a RETURN statement.

```
13900 TS$(SF) = NP$(SF) : GOSUB 300
```

The search token is stored in a temporary location (TS$(SF)). If the token was not stored, the token would be lost the first time the file is read. The program then jumps to the subroutine that begins at line 300.

Function of Subroutine At 300

The subroutine that begins at line 300 searches through the file, matching the token entered by the user, with the selected field from the record read off disk.

```
300 SUCC = 1
```

The successor record is set to 1, which points to the "DUMMY", header record.

```
310 PRINT D$;"READ ";F$;",R";SUCC
```

Line 310 instructs DOS to read from the successor record.

```
320 INPUT LK, CK$ : IF SUCC = 1 THEN 350
```

In line 320, the link pointer (LK), and the item are read from disk. If the successor pointer is equal to 1, the program branches to line 350. When SUCC = 1, it is pointing to the "DUMMY" header record.

```
350 PRED = SUCC : SUCC = LK : IF LK THEN 310
```

Line 350 assigns the successor record value (SUCC) to the predecessor record (PRED). The link pointer value is assigned to the successor record.

If the link pointer (LK) value equals 0, the end of the file has been reached. If the link pointer value is greater than 0, it means there is another record to be checked.

In line 320, IF SUCC = 1 THEN 350, is FALSE, the program defaults to line 330.

```
330 GOSUB 1000 : GOSUB 240 : IF TS$(SF) = LEFT$(NP$(SF),TL) THEN DF = 1 :
  RETURN
```

The subroutine that begins at line 1000 unpacks the item in CK$, and places it into the storage locations (NP$) to hold a record—see Function of Subroutine at 1000.

Subroutine 240 fills the NP$ fields with spaces—see Function of Subroutine at 240.

The third statement in line 330 compares the search token with the leftmost characters of NP$ of the search field (SF). It compares them so they are equal lengths (equal to the length of the search token). For example, a search token of "SMITH" will match the field value of "SMITHE", but a search token of "SMITHE" will not match a field value of "SMITH". The token "SMITH" will find any name beginning with "SMITH", such as "SMITHWICH", "SMITHWELL", etc. When the match is equal, the delete flag is set to 1 (DF = 1), and the program RETURNs to line 14200.

```
14200 PRINT D$;"PR#0" : PRINT D$;"IN#0" : IF DF = 0 THEN 16000
```

The first two statements in line 14200 tell DOS to send all PRINT statements to the CRT, and all input is to come from the keyboard. IF DF = 0 THEN 16000 causes the search to be terminated.

Lines 16000 through 16300 relate the interactive question, "TRY ANOTHER DELETION?, as to whether the user wants to delete another record.

IF DF = 0 THEN 16000 (line 14200) is FALSE (DF = 1), the program defaults to line 15100, which is GOSUB 400.

Function of Subroutine At 400

Line 400 clears the screen, and prints the literal "DELETE THIS RECORD ?" (AC$ = DELETE) on the screen.

```
440 FOR J = 1 TO NF : PRINT FD$(J);" = ";NP$(J) : PRINT
450 NEXT J : PRINT
```

Lines 440 and 450 print all the fields of the record, with their field descriptions, and field value, on the CRT.

```
460 INPUT "ENTER (Y OR N) ?";QU$
```

Line 460 asks for a Y, or N, answer to the question, "DELETE THIS RECORD ?". The user makes the decision to delete, or not delete, the record, and the program RETURNS to line 15400.

```
15400 IF QU$ <> "Y" THEN 15800
```

Line 15400 checks the user's response (line 460) to the question "DELETE THIS RECORD ?". If the response is NOT Y, the record is not deleted, and the program branches to line 15800.

If the response is Y, delete the record, the program defaults to line 15700 to GOSUB 700, to actually carry out the deletion process.

Function of Subroutine At 700

```
700 NODE = SUCC
```

The value of the successor (SUCC) record is stored in the record NODE.

```
710 PRINT D$;"WRITE ";F$;",R";PRED : GOSUB 150 : PRINT LK$
```

Line 700 tells DOS to write the predecessor (PRED) record. The subroutine at line 150 packs the link pointer (LK) into a string representation of four digits—see Function of Subroutine At 150. PRINT LK$ changes the

pointer at the predecessor (PRED) record to point to the record that the successor record points to. For example, the predecessor is record #10. The predecessor record points to the successor record. If the successor is record #20, and the successor record points to record #5, and record #20 is deleted, then the predecessor record is changed to point to record #5. NODE is set to record #20, because the NODE is the record to be deleted. The simple act of changing the value in the pointer causes the record to be deleted. This is one of the advantages of the linked list structure. Line 720 causes the program to RETURN to the second statement in line 15700 (TMP = LK).

This statement causes the link pointer (LK) value to be stored in a temporary (TMP) storage location. This is necessary because the NODE is to be placed on the available record list (AR), which is a linked list structure. The subroutine at line 800 is used to accomplish adding the vacant record to the linked list, and will change the value of the link pointer (LK).

Function of Subroutine At 800

```
800 LK = AR : GOSUB 150 : PRINT D$;"WRITE ";F$;",R";NODE : PRINT LK$ : AR
= NODE : NR = NR - 1
```

The available record pointer (AR) value is assigned to the link pointer (LK). The subroutine at line 150 is called to pack LK into a four digit string representation (LK$)—see Function of Subroutine At 150.

The value stored in LK$ is written into the record NODE by the third statement in line 800. NODE replaces the value stored in the available record pointer (AR), and the number of records (NR) is decremented because of the deleted record.

```
810 PRINT D$;"WRITE ";F$;",R0" : PRINT AR;",";NR : RETURN
```

Record 0 is updated to contain the new value in the available record pointer (AR), and the number of records (NR). The program RETURNS to the fourth statement in line 15700.

For example, if AR = 0 (meaning there are no vacant records in the file), and NODE equals 20 (NODE = 20), then "0000" is written into the RECORD #20, and AR now equals #20 (AR = 20). The next time a record is added, it will be placed in record #20, Fig. 11-3.

If AR is pointing to record #20, and record #15 is deleted, then record #15 will point to record #20, and AR will become #15 (AR = 15), Fig. 11-4.

The fourth statement in line 15700 is LK = TMP. The value assigned to LK is returned from its temporary storage location (TMP).

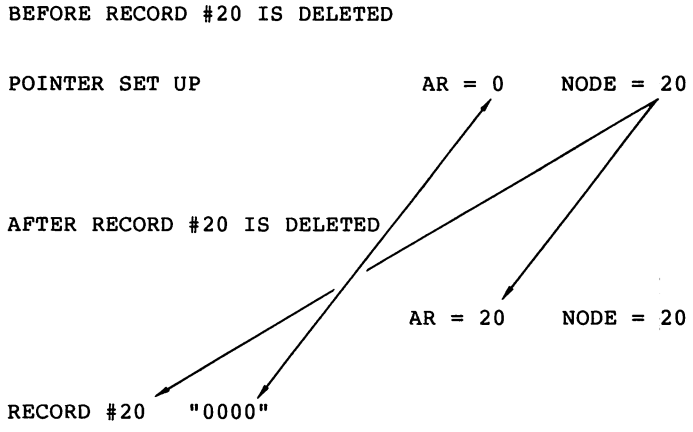


Fig. 11-3. No vacancies in the file.

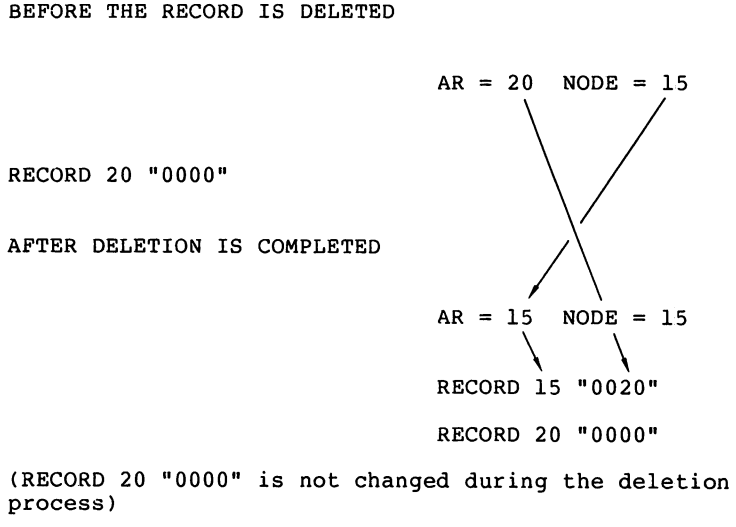


Fig. 11-4. Vacancies in the field—record No. 15 to be deleted.

If LK = 0, the end of the list has been reached, and the search routine is discontinued.

If LK is not equal to 0, the search is continued, so multiple deletions can be done on a file by using the same token. One token will delete all matching fields from the file, if so instructed.

If LK = 0, and the end of the list is reached, the interactive question in lines 16000 through 16300 asks the user if another deletion is to be performed.

```
15800 DF = 0 : SUCC = PRED : GOSUB 350 : GOTO 14200
```

DF = 0—the delete flag is set to 0, so that if the subroutine at line 300 RETURNs, and the delete flag equals 1, there is another record in the file that matches the token.

SUCC = PRED—after a deletion has been made, the successor pointer is incorrect, and the mistake must be corrected. To make the correction, SUCC is assigned the value stored in the predecessor record (PRED).

GOSUB 350—the subroutine that begins at line 350 will update the predecessor (PRED), and successor (SUCC), records, and then continues reading the rest of the file.

GOTO 14200—the program branches to line 14200, to determine if the search subroutine has found a record with the token matching the appropriate field of the record.

```
15900 PRINT D$;"PR#0" : PRINT D$;"IN#0"
```

Line 15900 tells DOS that PRINT statements are to go to the CRT, and all input is to come from the keyboard. This statement should be placed in a program any time the direction of the input and output is changed, so DOS will not be confused. The program falls through the interactive question. If the user discontinues the DELETE ITEMS option, the program RETURNs to line 24800, which is IF SE <> 6 THEN GOTO 21800. Since option No. 2, DELETE ITEMS, was selected, the menu is printed on the screen.

OPTION NO. 3, ALTER ITEMS

When option No. 3, ALTER ITEMS, is selected, the program branches to the subroutine that begins at line 7000.

Function of Subroutine At 7000

Line 7000 clears the screen, and prints, "ALTER ITEMS". The action string (AC$) is assigned the literal value "ALTER".

```
7030 DF = 0 : GOSUB 950 : TS$(SF) = NP$(SF) : IF TL = 0
THEN 7230
```

DF = 0—the delete flag is set to 0, which means no match between the token and the record field was found during the search. If DF = 1, it means a match was found between the token and the record field. GOSUB 950—this subroutine allows the user to enter the field on which to search, and also allows the user to enter the search token—see Function of Subroutine At 950.

TS$(SF) = NP$(SF)—the search token is stored temporarily. If the token length is 0, the ALTER ITEM function will be terminated. If TL = 0, the program branches to line 7230, to change the direction of the PRINT statements, and the input, and ask the user if "ANOTHER ALTER SEARCH?" is to be conducted.

```
7060 GOSUB 300
```

The subroutine that begins at line 300 searches the file to match the token against the selected field of the record—see Function of Subroutine At 300.

```
7090 IF DF = 0 THEN 7230
```

If no match was found during the processing in subroutine 300, the delete flag is still set to 0, and the program branches to line 7230, to ask the user, "ANOTHER ALTER SEARCH?".

If DF = 1, the program defaults to line 7180, to change the PRINT, and input directions to DOS.

```
7180 PRINT D$;"PR#0" : PRINT D$;"IN#0" : GOSUB 400 : IF QU$ <> "Y" THEN
     7220
```

GOSUB 400—the subroutine at line 400 displays the record, and asks the user whether any field in the record needs to be altered—see Function of Subroutine At 400.

If the answer to the question "ALTER THIS RECORD?" is not Y, the program branches to line 7220 to continue searching the file.

If the answer to the question "ALTER THIS RECORD?" is Y, the program defaults to line 7210, which is GOSUB 5000.

Function of Subroutine At 5000

The purpose of the subroutine that begins at line 5000 is to allow the fields of the record to be altered. It then changes the position of this record in the file, if an alteration is made that requires this action to be taken. Changing the field values of the record may make the ITEM less than the predecessor record, or greater than the successor record. For example, if the last name of "JONES" is changed to "SMITH", and the successor record has a last name of "NORMAN", then the altered record must be linked into the file after the successor record.

```
5000 FI = 0 : LO = 1 : HI = NF : HOME : PRINT "ENTER NEW VALUE OR PRESS
     'RETURN' " : PRINT "TO LEAVE VALUE UNCHANGED !"
```

FI = 0—flag input equals 0 means the record is being altered and only change the field value if anything is entered when a new field value is called

for. LO = 1 determines the first field in the record, and is used for the low index of a loop. HI = NF is used as the high index of a loop. The screen is cleared and "ENTER NEW VALUE OR PRESS 'RETURN' TO LEAVE VALUE UNCHANGED" is printed on the screen.

```
5030 T2 = TL : TMP$ = NP$(SF) : TMP = LK : T1 = PRED
```

T2 = TL—the temporary storage of the token length is assigned to the variable, T2. TMP$ = NP$(SF)—the search token is stored in a temporary string. TMP = LK—the link pointer is stored in a temporary location. T1 = PRED—the predecessor record number is stored in a temporary location, T1. The token length, token, and the predecessor record are all stored in temporary locations, because these same locations will be altered when checking to determine if the record should be relinked somewhere else in the file.

```
5050 GOSUB 500
```

The subroutine that begins at line 500 allows the user to enter the altered field values—see Function of Subroutine At 500.

```
5060 DF = 0 : GOSUB 6000 : IF DF = 0 THEN 5200
```

The delete flag (DF = 0) will now be used to indicate whether or not this record needs to be relinked in another position in the file. The program then jumps to the subroutine at 6000.

Function of Subroutine At 6000

The subroutine that begins at line 6000 checks the ITEM, and the predecessor record. If the altered record is less than the predecessor record, the program sets the delete flag to 1 (DF = 1), and RETURNS to the third statement in line 5060. If the altered record is greater than the predecessor record, the altered record is compared to the successor record. If the condition is TRUE (TR$ < RS$), the delete flag is set to 1 (DF = 1), and the program RETURNs to the third statement in line 5060.

```
6000 GOSUB 200 : GOSUB 240 : TR$ = NP$(1) : FOR J = 2 TO NF : TR$ = TR$ +
     NP$(J) : NEXT J
```

The subroutine that begins at line 200 stores the altered record in a temporary string (TS$)—see Function of Subroutine At 200. The subroutine that begins at line 240 packs the altered field in NP$ to their full field lengths—see Function of Subroutine At 240.

TR$ = NP$(1)—the first field in NP$ is stored in the temporary record. The loop then concatenates the other eight fields onto the temporary record string.

```
6020 IF PRED = 1 THEN 6065
```

If the predecessor record equals 1 (PRED = 1), the altered record is the first record in the file. In this case, there is no predecessor record to check the altered record against. The program then branches to line 6065.

If PRED = 1 is FALSE, the program defaults to line 6030.

```
6030 PRINT D$;"READ ";F$;",R";PRED : INPUT LK$,CK$
```

DOS reads the ITEM in the predecessor record, and inputs the link pointer value into LK$, and puts the record item into CK$.

```
6040 GOSUB 1000 : GOSUB 240
```

The subroutine that begins at line 1000 breaks apart CK$ into the NP$ field locations—see Function of Subroutine At 1000.

The subroutine that begins at line 240 fills the fields with spaces in NP$—see Function of Subroutine At 240.

```
6050 RS$ = NP$(1) : FOR J = 2 TO NF : RS$ = RS$ + NP$(J):NEXT J
```

The first field NP$(1) is assigned to the record storage string (RS$). The loop concatenates the other eight fields of the record. The total record is assigned to the temporary storage string (TR$). This assignment is done so the predecessor record can be compared to the successor record.

```
6060 IF TR$ < RS$ THEN DF = 1 : RETURN
```

If the altered record (TR$) is less than the previous record, the delete flag is set to 1 (DF = 1), and the program RETURNs to the third statement in line 5060.

If DF = 0 THEN 5200 means the delete flag was not set to 1, so the altered record remains at its position in the file.

Subroutine At 6000 Continues

If line 6060 (IF TR$ < RS$ THEN DF = 1) is FALSE, the program defaults to line 6056.

```
6050 IF LK = 0 THEN RETURN
```

If the altered record is the last record in the file, line 6065 is TRUE, and the altered record does not need to be relinked in the file.

If line 6065 is FALSE, the program defaults to line 6070, which tells DOS to read the record at the link pointer (LK).

```
6070 PRINT D$;"READ ";F$;",R";LK
6080 INPUT LK$,CK$ : GOSUB 1000 : GOSUB 240
```

The link pointer and the ITEM (CK$) are read from disk.

GOSUB 1000 breaks the ITEM (CK$) into nine separate fields and places them in NP$—see Function of Subroutine At 1000.

GOSUB 240 fills all the fields in NP$ with necessary spaces—see Function of Subroutine At 240.

```
6090 RS$ = NP$(1) : FOR J = 2 TO NF : RS$ = RS$ + NP$(J) : NEXT J
```

The first field in NP$(1) is assigned to the storage record string (RS$). The loop adds the other eight fields onto RS$.

```
6100 IF TR$ > RS$ THEN DF = 1
```

If the altered ITEM in TR$ is greater than the ITEM in the record string (RS$) that follows it in the file, then the delete flag is set to 1 (DF = 1), and the program RETURNs to the third statement in line 5060. The third statement in line 5060 is IF DF = 0 THEN 5200. If DF = 0, the program branches to line 5200.

```
5200 LK = TMP : GOSUB 150 : PRINT D$;"WRITE ";F$;",R";SUCC : PRINT LK$ :
     GOSUB 6500
```

The line at 5200 writes the altered record back to the file in its new form. The link pointer is returned from its temporary location. The subroutine that begins at line 150 creates a four digit string ("0000") representation in link pointer string (LK$).

PRINT D$;"WRITE ";F$;",R";SUCC tells DOS to write the altered record at the successor record (the same place in the file where the record was found).

PRINT LK$ prints LK$ to disk, and then jumps to the subroutine that begins at line 6500.

Function of Subroutine At 6500

The subroutine that begins at line 6500 takes the nine temporary storage locations in TS$, and removes the trailing spaces in each field so the fields can be put back on disk in their packed form.

```
6500 FOR J = 1 TO NF
6510 FOR K = FL(J) TO 1 STEP -1
```

```
6520 IF MID$(TS$(J),K,1) <> " " THEN TS$(J) = LEFT$(TS$(J),K) : GOTO 6700
6530 NEXT K
6550 TS$(J) = ""
6700 NEXT J
6800 FOR J = 1 TO NF - 1
6850 PRINT TS$(J) ;CHR$(18);
6900 NEXT J : PRINT TS$(NF) : RETURN
```

The J loop variable is set to examine all nine fields of the record. The K loop variable is set to examine the character in the field starting with the rightmost character, and terminating with the first nonspace character.

Line 6520, and the doubly nested loops, examine the characters in each field starting with the rightmost character. If the character is blank, the statement is FALSE, and the program defaults to line 6530 (NEXT K). If the character is a nonblank character, the statement is TRUE, and the characters from the last check, to the beginning of the field (No. 1 character in the field), are assigned to TS$(J), and the program branches to line 6700 (NEXT J).

If the K loop executes, and the check at line 6520 is always FALSE, the field contains all blanks, and the statement at line 6550 sets the field to a null value (TS$(J) = "").

When all the fields have all the blanks removed, the loop (lines 6800 through 6900) prints the first eight fields in TS$ (in packed form), and each field is followed by a CHR$(18). The ninth field is then printed to disk, and the program RETURNs to line 5230.

```
5230 LK = TMP
```

The link pointer that has been stored in a temporary location (TMP) is assigned to the link pointer.

```
5240 TL = T2 : NP$(SF) = TMP$ : RETURN
```

Line 5240 returns the token length and token from the search string to their original memory locations. The token length and the token were stored in temporary locations in line 5030, so they would not be lost during the search. The program RETURNs to line 7220.

```
5090 GOSUB 700
```

The subroutine that begins at line 700 changes the link pointers so the predecessor record points to the altered record's successor—see Function of Subroutine At 700.

```
5100 GOSUB 4000
```

The subroutine that begins at line 4000 finds the new position in the file where the altered record should be linked—see Function of Subroutine At 4000.

```
5120 LK = NODE : GOSUB 150 : PRINT D$;"WRITE ";F$;",R";PRED : PRINT LK$
```

LK = NODE assigns the value stored in the NODE pointer to the link pointer (LK). GOSUB 150 converts the numeric value stored in the link pointer (LK) to a four digit string representation. The link pointer is converted to a string representation to make sure it always contains four digits—see Function of Subroutine At 150.

The link pointer is then written to the predecessor (PRED) record.

```
5140 LK = SUCC : GOSUB 150 : PRINT D$;"WRITE ";F$;",R";NODE : PRINT LK$ :
     GOSUB 6500
```

The successor record pointer (SUCC) is assigned to the link pointer (LK). The subroutine at line 150 converts the numeric value in the link pointer to a four digit string representation. The successor pointer (SUCC) is written to the record pointer (NODE), and the link pointer string (LK$) is written to disk. The program jumps to the subroutine at line 6500.

The subroutine that begins at line 6500 removes trailing spaces from each of the fields of the ITEM, and writes the record in packed form on the disk—see Function of Subroutine At 6500.

```
5160 SUCC = T1 : GOTO 5230
```

Line 5160 takes the predecessor pointer (line 5030) stored in the variable, T1, and assigns it to the successor pointer (SUCC). The program jumps to line 5230 and 5240 to restore the other three values, TMP, token length, and the token, which have been stored in temporary locations. The program RETURNs to line 7220.

```
7220 DF = 0 : GOSUB 350 : GOTO 7090
```

The delete flag is set off (DF = 0), and the program jumps to the subroutine at line 350, to continue examining records. The program jumps to line 7090 to see if the search has found a match with the token.

```
7230 PRINT D$;"PR#0" : PRINT D$;"IN#0" : PRINT "ALTER SEARCH
     TERMINATED"
```

When the search is terminated, the program has branched from line 7090 (IF DF = 0 THEN 7230).

Line 7230 sets up the input and PRINT directions from the keyboard and

CRT, and asks the user to answer the interactive question "ANOTHER ALTER SEARCH?", and the program RETURNs to line 24800.

```
24800 IF SE <> 6 THEN GOTO 21800
```

Since option No. 3, ALTER ITEMS, was selected, the program branches to line 21800 to print out the menu.

OPTION NO. 4, PRINT THE LIST

If option No. 4, "PRINT THE LIST", is selected, the program branches to the subroutine at line 9000.

Function of Subroutine At 9000

Line 9000 clears the screen, and prints PRINT THE FILE. Then the program asks the user if the program should stop after every record, or should print the entire list without stopping.

```
9010 PRINT D$;"READ ";F$;",R1":INPUT PRED: IF PRED = 0 THEN PRINT
     D$;"PR#0":PRINT D$;"IN#0": GOTO 9280
```

In line 9010, Record #1 is read. The predecessor pointer (PRED) is read from the disk. The predecessor pointer is the value of the first record number in the file. If the file is empty, (PRED = 0), the program branches to line 9280 to RETURN to print the menu on the screen.

```
9020 PRINT D$;"READ ";F$;",R";PRED
```

Line 9020 Sets DOS to read the record stored at the predecessor record (PRED).

```
9040 INPUT SUCC,CK$ : PRINT D$;"PR#0" : PRINT D$;"IN#0"
```

Line 9040 inputs the link pointer to the successor pointer, and the PRINT and input directions are set to CRT and the keyboard.

```
9100 GOSUB 1000
```

The subroutine that begins at line 1000 breaks CK$ apart into nine fields, storing them in NP$—see Function of Subroutine At 1000.

```
9180 FOR J = 1 TO NF
9200 PRINT FD$(J);" = ";NP$(J)
9220 NEXT J
```

The loop in lines 9180 through 9220 writes the nine field descriptors, and

the data stored in the field (value of the associated field). The ITEM data is written on the screen, one record at a time.

```
9230 IF QU$ = "N" THEN 9260
```

If the answer to the interactive question is N, for no, the statement that stops the printout after each record is bypassed, and the entire list is printed out without stopping.

```
9240 PRINT : INPUT "ENTER Q TO QUIT ELSE PRESS RETURN !";Q$
     : IF Q$ = "Q" THEN 9280
```

If the answer to the interactive question is Q, for quit, the program branches to line 9280, to RETURN to line 24800, and prints out the menu.

If the answer is RETURN, the program defaults to line 9260.

```
9260 IF SUCC > 0 THEN PRED = SUCC : GOTO 9020
```

If the successor pointer is greater than zero (SUCC > 0), the successor pointer is stored in the predecessor pointer, and the program jumps to line 9020 to get the next record. The program RETURNs to line 24800 (IF SE < > 6 THEN GOTO 21800). Since option No. 4, "PRINT THE LIST", was selected, the program branches to line 21800 to print out the menu.

OPTION NO. 5, EXAMINE THE FIELDS

If option No. 5, "EXAMINE THE ";NF;" FIELDS", is selected, the program branches to the subroutine at line 16900.

Function of Subroutine At 16900

Line 16900 clears the screen, and prints FIELD CLARIFICATION, on the screen.

```
17200 PRINT : PRINT "FIELD #";TAB(10);"FIELD
      NAME";TAB(35);"LENGTH" : PRINT : PRINT
```

Line 17200 places "FIELD #", starting in column #1, "FIELD NAME", starting in column #10, and the maximum field length starts in column #35.

```
17500 FOR J = 1 TO NF
17800 PRINT J; TAB (10); FD$(J); TAB (35); FL(J)
18100 NEXT J
```

The loop prints the nine fields, field descriptors, and their maximum length on the screen.

```
18400 PRINT : PRINT : PRINT "TOTAL RECORD LENGTH IS ";RL
```

Line 18400 prints the value of the total record length. This information is for user references purposes only. In line 32000, if NF was changed to 12 (NF = 12), and the associated data field descriptors and field lengths were entered, the calculations in subroutine at 16900 would automatically reflect these changes.

The Random Access File Program was designed to input the record data from the keyboard, store the record on disk, and output the records to the CRT.

We are using a TRS-80, DWP-410 printer, interfaced to an Apple II computer, through a R-611C parallel printer interface card. The interface card was manufactured by MICROTEC, Inc., 9514 Chesapeake Drive, San Diego, California, 92123. The DWP-410 has the same pin configuration as the TEXAS INSTRUMENTS 810/820, or CENTRONICS 702/703/779/353/152.

To produce Figs. 11-5, and 11-6, output to the printer, the program must be modified. The statement (PRINT D$;"PR#0" : PRINT D$;"IN#0") that changes direction from DOS to the CRT, and keyboard, also turns off the printer (PR#0).

To correct this deficiency, and leave the printer on, the PR#0 statement must be changed in the program.

```
1. PRINT D$;"PR#0"
2. PRINT D$;"PR#1"
```

Statement No. 1 must be replaced by statement No. 2 in program lines, 7180, 7230, 9010, 9040, 11800, 14200, 15900, 30030, and 31250.

Fig. 11-5 is a RUN of the Random Access File Program, in which 5 records are input.

Fig. 11-6 is a RUN of the Random Access File Program in which options No. 2, No. 3, No. 4, No. 5, and No. 6 are exercised.

Fig. 11-6, option No. 2, "DELETE ITEMS", deleted the record of WILLIAM JENNINGS of CLUTE, TX. The search token was "CLU", for "CLUTE". When the "CLUTE" field was found, Y, for yes, was entered, RETURN was pressed, and the record was deleted.

The pointers, Fig. 11-6, link the file in the following manner. "RECORD 0006" links to "RECORD 0002", which is the record to be deleted, and "RECORD 0002" LINKS TO "RECORD 0005".

After the deletion, Fig. 11-6, "RECORD 0006" now points to "RECORD 0005". "RECORD 0002" is placed on the available record list as a vacant record in the file. The available record pointer ("RECORD 0000")

```
 MON C,I,O

]RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)?


 ENTER FILE NAME ?RTEST

!!!THE NAME IS GOOD
DO YOU WANT TO BUILD A NEW FILE

OR USE AN EXISTING FILE???


ENTER N(NEW) OR O(OLD)? N
OPEN RTEST,L130,S6,D1
WRITE RTEST,R0
0,1
WRITE RTEST,R1
0000,

IN#0
PR#1

SELECT FUNCTION FROM FOLLOWING LIST


1. ADD ITEMS

2. DELETE ITEMS

3. ALTER ITEMS

4. PRINT THE LIST

5. EXAMINE THE 9 FIELDS

6. CLOSE THE FILE AND END


ENTER REQUEST ? 1

               ADD RECORDS

LASTNAME = ?JENNINGS
```

Fig. 11-5. RUN of random access program.

```
FIRSTNAME = ?WILLIAM

STREET = ?827 MCFADDEN

CITY = ?CLUTE

STATE = ?TX

ZIP CODE = ?74301

PHONE NUMBER = ?713-227-8642

PURCHASE DATE = ?10/11/81

MEMO = ?AMERICAN EXPRESS
WRITE RTEST,R0
0,2
READ RTEST,R1
?0000,
WRITE RTEST,R1
0002
WRITE RTEST,R2
0000
JENNINGSWILLIAM827 MCFADDENCLUTETX74301713-227-86421
0/11/81AMERICAN EXPRESS

PR#1

IN#0
DISCONTINUE ADDING ITEMS ?

LASTNAME = ?PENDLETON

FIRSTNAME = ?JOHN

STREET = ?846 GOLOID RD.

CITY = ?DALLAS

STATE = ?KY

ZIP CODE = ?47713

PHONE NUMBER = ?612-478-8900

PURCHASE DATE = ?14/6/82

MEMO = ?KENTUCKY NAL'T BANK
WRITE RTEST,R0
```

Fig. 11-5—*cont.* RUN of random access program.

```
0,3
READ RTEST,R1
?0002
??
READ RTEST,R2
?0000
??JENNINGSWILLIAM827 MCFADDENCLUTETX74301713-227-864
210/11/81AMERICAN EXPRESS

WRITE RTEST,R2
0003
WRITE RTEST,R3
0000
PENDLETONJOHN846 GOLOID RD.DALLASKY47713612-478-8900
14/6/82KENTUCKY NAL'T BANK

PR#1

IN#0
DISCONTINUE ADDING ITEMS ?

LASTNAME = ?SMITH

FIRSTNAME = ?CHARLES

STREET = ?890 W. SALLIE

CITY = ?BIGFOOT

STATE = ?WY

ZIP CODE = ?27081

PHONE NUMBER = ?307-682-7711

PURCHASE DATE = ?17/5/82

MEMO = ?LARIME NAT'L BANK
WRITE RTEST,R0
0,4
READ RTEST,R1
?0002
??
READ RTEST,R2
?0003
??JENNINGSWILLIAM827 MCFADDENCLUTETX74301713-227-864
210/11/81AMERICAN EXPRESS
```

Fig. 11-5—*cont.* RUN of random access program.

```
READ RTEST,R3
?0000
??PENDLETONJOHN846 GOLOID RD.DALLASKY47713612-478-89
0014/6/82KENTUCKY NAL'T BANK

WRITE RTEST,R3
0004
WRITE RTEST,R4
0000
SMITHCHARLES890 W. SALLIEBIGFOOTWY27081307-682-77111
7/5/82LARIME NAT'L BANK

PR#1

IN#0
DISCONTINUE ADDING ITEMS ?

LASTNAME = ?MOORE

FIRSTNAME = ?TED

STREET = ?2382 ASHBY

CITY = ?SLIDELL

STATE = ?LA

ZIP CODE = ?62311

PHONE NUMBER = ?417-299-7833

PURCHASE DATE = ?13/4/81

MEMO = ?SLIDELL NAT'L BANK
WRITE RTEST,R0
0,5
READ RTEST,R1
?0002
??
READ RTEST,R2
?0003
??JENNINGSWILLIAM827 MCFADDENCLUTETX74301713-227-864
210/11/81AMERICAN EXPRESS

READ RTEST,R3
?0004
??PENDLETONJOHN846 GOLOID RD.DALLASKY47713612-478-89
0014/6/82KENTUCKY NAL'T BANK
```

Fig. 11-5—*cont.* RUN of random access program.

```
WRITE RTEST,R2
0005
WRITE RTEST,R5
0003
MOORETED2382 ASHBYSLIDELLLA62311417-299-783313/4/81S
LIDELL NAT'L BANK

PR#1

IN#0
DISCONTINUE ADDING ITEMS ?

LASTNAME = ?CURE

FIRSTNAME = ?DALE

STREET = ?5026 FANNIN

CITY = ?NEDERLAND

STATE = ?TX

ZIP CODE = ?76302

PHONE NUMBER = ?713-755-9419

PURCHASE DATE = ?10/03/81

MEMO = ?1ST NAT'L OF TX
WRITE RTEST,R0
0,6
READ RTEST,R1
?0002
??
READ RTEST,R2
?0005
??JENNINGSWILLIAM827 MCFADDENCLUTETX74301713-227-
  864210/11/81AMERICAN EXPRESS
WRITE RTEST,R1
0006
WRITE RTEST,R6
0002
CUREDALE5026 FANNINNEDERLANDTX76302713-755-941910/03/81
  1ST NAT'L OF TX
PR#1

IN#0
DISCONTINUE ADDING ITEMS ?Y
SELECT FUNCTION FROM FOLLOWING LIST
```

Fig. 11-5—*cont.* RUN of random access program.

```
1. ADD ITEMS

2. DELETE ITEMS

3. ALTER ITEMS

4. PRINT THE LIST

5. EXAMINE THE 9 FIELDS

6. CLOSE THE FILE AND END


ENTER REQUEST ? 6
WRITE RTEST,R0
0,6
CLOSE
LOCK RTEST,S6,D1

]
```

Fig. 11-5—*cont.* RUN of random access program.

is written to "RECORD 0002", and the available record pointer is now "RECORD 0002".

The rest of the records in the file are searched, and no other match is found with the token CLU.

Option No. 3, ALTER ITEMS, Fig. 11-6, selects field #6, ZIP CODE, for the search. The token, 47713, was entered as the zip code for the search. The word NAL'T had been misspelled in field #9, MEMO. When Y for yes was entered in answer to the question "ALTER THIS RECORD?", the nine fields for possible alteration were printed. RETURN was pressed for each field, until field #9, MEMO, was reached.

The correct spelling, KENTUCKY NAT'L BANK, was entered into field #9, and RETURN was pressed.

The "ALTER ITEMS" option was ended by typing N, for no, in answer to the question "ANOTHER ALTER SEARCH?".

The menu was printed, and option No. 4, "PRINT THE LIST", was selected, Fig. 11-6. The option to stop printing the list after each record was exercised. When the four record list (one record was deleted) was output, the program RETURNed to the menu.

Option No. 5, "EXAMINE THE 9 FIELDS", was selected, Fig. 11-6, and the field number, field name, and field length was output. In the program, the field length is TABed to begin in column #35. The TAB function

```
 RUN
UNIT ACCESSED IS SLOT 6 DRIVE 1

CHANGE THESE VALUES(Y OR N)?


 ENTER FILE NAME ?RTEST

!!!THE NAME IS GOOD
DO YOU WANT TO BUILD A NEW FILE

OR USE AN EXISTING FILE???


ENTER N(NEW) OR O(OLD)? O
UNLOCK RTEST,S6,D1
OPEN RTEST,L130,S6,D1
READ RTEST,R0
?0,6
PR#1

IN#0
SELECT FUNCTION FROM FOLLOWING LIST



1. ADD ITEMS

2. DELETE ITEMS

3. ALTER ITEMS

4. PRINT THE LIST

5. EXAMINE THE 9 FIELDS

6. CLOSE THE FILE AND END


ENTER REQUEST ? 2

              DELETE ITEMS
ENTER A FIELD TO SEARCH ON (1-9) !


FIELD =?4
ENTER A TOKEN TO SEARCH ON !
```

Fig. 11-6. RUN of random access program.

```
CITY = ?CLU
READ RTEST,R1
?0006
??
READ RTEST,R6
?0002
??CUREDALE5026 FANNINNEDERLANDTX76302713-755-941910/
03/811ST NAT'L OF TX

READ RTEST,R2
?0005
??JENNINGSWILLIAM827 MCFADDENCLUTETX74301713-227-864
210/11/81AMERICAN EXPRESS

PR#1

IN#0
DELETE THIS RECORD ?


LASTNAME = JENNINGS

FIRSTNAME = WILLIAM

STREET = 827 MCFADDEN

CITY = CLUTE

STATE = TX

ZIP CODE = 74301

PHONE NUMBER = 713-227-8642

PURCHASE DATE = 10/11/81

MEMO = AMERICAN EXPRESS


ENTER (Y OR N) ?Y
WRITE RTEST,R6
0005
WRITE RTEST,R2
0000
WRITE RTEST,R0
2,5
READ RTEST,R5
?0003
```

Fig. 11-6—*cont.* RUN of random access program.

```
??MOORETED2382 ASHBYSLIDELLLA62311417-299-783313/4/8
1SLIDELL NAT'L BANK

READ RTEST,R3
?0004
??PENDLETONJOHN846 GOLOID RD.DALLASKY47713612-478-89
0014/6/82KENTUCKY NAL'T BANK

READ RTEST,R4
?0000
??SMITHCHARLES890 W. SALLIEBIGFOOTWY27081307-682-771
117/5/82LARIME NAT'L BANK

PR#1

IN#0
DELETION ON SEARCH TERMINATED


TRY ANOTHER DELETION ?N
SELECT FUNCTION FROM FOLLOWING LIST


1. ADD ITEMS

2. DELETE ITEMS

3. ALTER ITEMS

4. PRINT THE LIST

5. EXAMINE THE 9 FIELDS

6. CLOSE THE FILE AND END


ENTER REQUEST ? 3
              ALTER ITEMS
ENTER A FIELD TO SEARCH ON (1-9) !


FIELD =?6
ENTER A TOKEN TO SEARCH ON !


ZIP CODE = ?47713
READ RTEST,R1
```

Fig. 11-6—*cont.* RUN of random access program.

```
?0006
??
READ RTEST,R6
?0005
??CUREDALE5026 FANNINNEDERLANDTX76302713-755-941910/
03/811ST NAT'L OF TX

READ RTEST,R5
?0003
??MOORETED2382 ASHBYSLIDELLLA62311417-299-783313/4/8
1SLIDELL NAT'L BANK

READ RTEST,R3
?0004
??PENDLETONJOHN846 GOLOID RD.DALLASKY47713612-478-89
0014/6/82KENTUCKY NAL'T BANK

PR#1

IN#0
ALTER THIS RECORD ?


LASTNAME = PENDLETON

FIRSTNAME = JOHN

STREET = 846 GOLOID RD.

CITY = DALLAS

STATE = KY

ZIP CODE = 47713

PHONE NUMBER = 612-478-8900

PURCHASE DATE = 14/6/82

MEMO = KENTUCKY NAL'T BANK


ENTER (Y OR N) ?Y

ENTER NEW VALUE OR PRESS 'RETURN'
TO LEAVE VALUE UNCHANGED !

LASTNAME = ?
```

Fig. 11-6—*cont.* RUN of random access program.

```
FIRSTNAME = ?

STREET = ?

CITY = ?

STATE = ?

ZIP CODE = ?

PHONE NUMBER = ?

PURCHASE DATE = ?

MEMO = ?KENTUCKY NAT'L BANK
READ RTEST,R5
?0003
??MOORETED2382 ASHBYSLIDELLLA62311417-299-783313/4/8
1SLIDELL NAT'L BANK

READ RTEST,R4
?0000
??SMITHCHARLES890 W. SALLIEBIGFOOTWY27081307-682-771
117/5/82LARIME NAT'L BANK

WRITE RTEST,R3
0004
PENDLETONJOHN846 GOLOID RD.DALLASKY47713612-478-8900
14/6/82KENTUCKY NAT'L BANK

READ RTEST,R4
?0000
??SMITHCHARLES890 W. SALLIEBIGFOOTWY27081307-682-771
117/5/82LARIME NAT'L BANK

PR#1

IN#0
ALTER SEARCH TERMINATED
ANOTHER ALTER SEARCH ? N
SELECT FUNCTION FROM FOLLOWING LIST


1. ADD ITEMS

2. DELETE ITEMS

3. ALTER ITEMS
```

Fig. 11-6—*cont.* RUN of random access program.

```
4. PRINT THE LIST

5. EXAMINE THE 9 FIELDS

6. CLOSE THE FILE AND END


ENTER REQUEST ? 4
          PRINT THE FILE
STOP AFTER EVERY RECORD (Y OR N) ? Y
READ RTEST,R1
?0006
READ RTEST,R6
?0005
??CUREDALE5026 FANNINNEDERLANDTX76302713-755-941910/
03/811ST NAT'L OF TX

PR#1

IN#0
LASTNAME = CURE
FIRSTNAME = DALE
STREET = 5026 FANNIN
CITY = NEDERLAND
STATE = TX
ZIP CODE = 76302
PHONE NUMBER = 713-755-9419
PURCHASE DATE = 10/03/81
MEMO = 1ST NAT'L OF TX

ENTER Q TO QUIT ELSE PRESS RETURN !
READ RTEST,R5
?0003
??MOORETED2382 ASHBYSLIDELLLA62311417-299-783313/4/8
1SLIDELL NAT'L BANK

PR#1

IN#0
LASTNAME = MOORE
FIRSTNAME = TED
STREET = 2382 ASHBY
CITY = SLIDELL
STATE = LA
ZIP CODE = 62311
PHONE NUMBER = 417-299-7833
PURCHASE DATE = 13/4/81
MEMO = SLIDELL NAT'L BANK
```

Fig. 11-6—*cont.* RUN of random access program.

```
ENTER Q TO QUIT ELSE PRESS RETURN !
READ RTEST,R3
?0004
??PENDLETONJOHN846 GOLOID RD.DALLASKY47713612-478-89
0014/6/82KENTUCKY NAT'L BANK

PR#1

IN#0
LASTNAME = PENDLETON
FIRSTNAME = JOHN
STREET = 846 GOLOID RD.
CITY = DALLAS
STATE = KY
ZIP CODE = 47713
PHONE NUMBER = 612-478-8900
PURCHASE DATE = 14/6/82
MEMO = KENTUCKY NAT'L BANK

ENTER Q TO QUIT ELSE PRESS RETURN !
READ RTEST,R4
?0000
??SMITHCHARLES890 W. SALLIEBIGFOOTWY27081307-682-771
117/5/82LARIME NAT'L BANK

PR#1

IN#0
LASTNAME = SMITH
FIRSTNAME = CHARLES
STREET = 890 W. SALLIE
CITY = BIGFOOT
STATE = WY
ZIP CODE = 27081
PHONE NUMBER = 307-682-7711
PURCHASE DATE = 17/5/82
MEMO = LARIME NAT'L BANK

ENTER Q TO QUIT ELSE PRESS RETURN !
SELECT FUNCTION FROM FOLLOWING LIST


1. ADD ITEMS

2. DELETE ITEMS

3. ALTER ITEMS
```

Fig. 11-6—*cont.* RUN of random access program.

```
4. PRINT THE LIST

5. EXAMINE THE 9 FIELDS

6. CLOSE THE FILE AND END


ENTER REQUEST ? 5
         FIELD CLARIFICATION

FIELD #          FIELD NAME                          LENGTH


1          LASTNAME                          12
2          FIRSTNAME                          8
3          STREET                            15
4          CITY                              10
5          STATE                              2
6          ZIP CODE                           5
7          PHONE NUMBER                                12
8          PURCHASE DATE                               8
9          MEMO                              44


TOTAL RECORD LENGTH IS 130

PRESS ANY KEY TO CONTINUE ...
                  SELECT FUNCTION FROM FOLLOWING LIST


1. ADD ITEMS

2. DELETE ITEMS

3. ALTER ITEMS

4. PRINT THE LIST

5. EXAMINE THE 9 FIELDS

6. CLOSE THE FILE AND END


ENTER REQUEST ? 6
WRITE RTEST,R0
2,5
CLOSE
LOCK RTEST,S6,D1
```

Fig. 11-6—*cont.* RUN of random access program.

to the CRT, and the printer is interpreted differently, and the printer does not TAB to column #35. The printer spaces 35 positions in relation to the end of the field name. When any key is pressed, the program prints out the main menu.

Option No. 6, "CLOSE THE FILE AND END", is selected, Fig. 11-6. "R0" contains the available record pointer (AR = 0002), and the number of records in the file (NR = 5—four records and the DUMMY). The file is CLOSEed, and LOCKed.

chapter 12

Executive Files

An executive file is a text file that contains a series of commands that the Apple II is to execute. An executive file is used to control a series of steps that do not require user intervention. The executive file is useful when a series of inputs is required to accomplish a goal. The executive file relieves the user from making keyboard inputs to do a specific task.

In copying a disk using the FID (disk copy utility) program from APPLE COMPUTER INC., multiple keyboard inputs are required. Those inputs are listed below.

1. The FID program must be RUN (BRUN FID).
2. A specific option must be input from the keyboard.
3. The value of the source slot must be input.
4. The value of the source drive must be input.
5. The value of the destination slot must be input.
6. The value of the destination drive must be input.
7. The file names of the files to be copied must be input.
8. A decision (Y or N) must be input in answer to the question, "DO YOU WANT PROMPTING?".
9. Any key must be pressed (EXCEPT ESCAPE), to begin the disk copy function.
10. After the disk is copied, any key must be pressed to return to the menu.

11. Another selection must be made to determine the next option. (In this case, option No. 9, is selected, to "QUIT", the program).

These 11 steps can be handled, without user intervention, by using an executive file.

To create an executive file, there is a sequence of events that must occur in the proper order.

1. Write the executive program to fulfill a specific need.
2. RUN the executive file program. The RUN, of the executive file program, creates a file with a specific name. (In this case, the name of the executive file is XCTEST).
3. XCTEST (file name) must be executed for it to accomplish its goal. EXEC XCTEST is the command that causes XCTEST, text file, to execute.

In summary, the executive file program is RUN to create an executive file with a specific name (in this case, XCTEST). EXEC XCTEST is the command that causes the executive file to perform a specific set of keyboard inputs to accomplish a task, or tasks.

The tasks that the executive file program accomplishes are as follows.

1. Open a file, XCTEST, using an Applesoft program.
2. Write to file, XCTEST.
3. Print a CALL to the monitor command.
4. Print a list page zero of Apple memory command.
5. Print a return to the Applesoft language command.
6. Print a RUN the executive header program command.
7. Print a BRUN FID (the APPLE utility disk copy program) command.
8. All the values that the FID program needs are written to the XCTEST file.
9. CLOSE the XCTEST file.

When the executive file program, Fig. 12-1, is RUN, it creates the XCTEST file, Fig. 12-2.

Figs. 12-1 and 12-2 have lines correspondingly marked. Line 10, Fig. 12-1, is missing from Fig. 12-2, because "D$ = CHR$(4)" is a control character and does not print out. Line #40, "CALL −151", is a command to get into the machine language section (monitor) of the Apple computer. Line #50, "0L", is the command to LIST PAGE ZERO. Line #60, "E000G", is the command to return to the Applesoft language. Line #70, "RUN CH 12 PROG-EXECD", runs a small header program. Line #80,

```
10 D$ =  CHR$ (4)
20  PRINT D$;"OPEN XCTEST"
30  PRINT D$;"WRITE XCTEST"
40  PRINT "CALL -151"
50  PRINT "0L"
60  PRINT "E000G"
70  PRINT "RUN CH 12 PROG-EXECD"
80  PRINT "BRUN FID"
90  PRINT "1": REM FID OPTION 1
100  PRINT "6": REM SOURCE SLOT
110  PRINT "1": REM SOURCE DRIVE
120  PRINT "6": REM DESTINATION SLOT
130  PRINT "2": REM DESTINATION DRIVE
140  PRINT "=": REM FILENAME
150  PRINT "N": REM NO PROMPTING
160  PRINT : REM  'RETURN'
170  PRINT "DUMY": REM CHANGE NAME FOR DUPLICATE HELLO
      PROGRAM
180  PRINT : REM RETURN AFTER COPY COMPLETE
190  PRINT "9": REM SELECTION OPTION TO END
300  PRINT D$;"CLOSE XCTEST"
310  END
```

Fig. 12-1. Executive file program.

```
10  REM  OPEN XCTEST--------------------LINE #20
20  REM  WRITE XCTEST-------------------LINE #30
30  REM  CALL -151----------------------LINE #40
40  REM  0L-----------------------------LINE #50
50  REM  E000G--------------------------LINE #60
60  REM  RUN CH 12 PROG-EXECD-----------LINE #70
62  REM  BRUN FID-----------------------LINE #80
64  REM  1------------------------------LINE #90
66  REM  6------------------------------LINE #100
68  REM  1------------------------------LINE #110
70  REM  6------------------------------LINE #120
72  REM  2------------------------------LINE #130
74  REM  =------------------------------LINE #140
76  REM  N------------------------------LINE #150
78  REM   ------------------------------LINE #160
80  REM  DUMY---------------------------LINE #170
82  REM   ------------------------------LINE #180
84  REM  9------------------------------LINE #190
86  REM  CLOSE XCTEST-------------------LINE #300
```

Fig. 12-2. RUN of the executive file program.

"BRUN FID", RUNs the disk copy program. Line #90, "PRINT "1", is the command that causes option No. 1, of the FID program, to be executed. Option No. 1 is "COPY FILES". Line #100 causes a 6 to be input as the source slot number. Line #110 causes number 1 to be input as the source disk drive number. Line #120 causes the number 6 to be input as the destination slot number. Line #130 causes the number 2 to be input as the destination disk drive number. Line #140, (=), is the wild card designator,

that causes all files on the disk to be copied. Line #150, "N", for no, is the response to the question, "DO YOU WANT PROMPTING?". Line 160 causes a RETURN to be generated, so the program will continue processing. Line #170 is the replacement name, DUMMY, to be used since the copied disk already has the file "HELLO" on it. Line #180 causes a RETURN to be input after the disk has been copied. Line #190 causes 9 to be input in response to the question "WHICH WOULD YOU LIKE?". Line #300 CLOSEs the XCTEST file.

Fig. 12-3 is the listing of a program which the XCTEST executive file runs. This does nothing but display a header and pause. It is intended to show that executive files may run several programs consecutively.

```
]LIST

10  HOME
20  VTAB 4: PRINT "THIS DISK WILL
      BE COPIED"
50  FOR J = 1 TO 5000
60  NEXT J

]PR#0
```

Fig. 12-3. Program that XC TEXT runs as a demonstration.

Fig. 12-4 was created when the command EXEC XCTEST was given. XCTEST was EXECuted on a three disk drive system, and the disk was in a drive whose interface card was in slot #4, drive #1.

In Fig. 12-4, page zero (0) of Apple memory was printed out. The phrase, "THIS DISK WILL BE COPIED", from Fig. 12-3, was printed. The correct responses from the XCTEST file caused the disk to be copied.

The banner of the FID program is distorted on the printout because the banner is displayed as a single line of 160 characters that wraps around to the next line after 40 characters on the CRT. This causes the display to appear as 4 lines. The printer is not limited to 40 characters so the display continues until the end of line on the printer is reached.

```
 EXEC XCTEST,S4,D1
]

*

0000-   4C 3C D4    JMP   $D43C
0003-   4C 3A DB    JMP   $DB3A
0006-   FF          ???
0007-   FF          ???
0008-   FF          ???
0009-   FF          ???
000A-   4C 99 E1    JMP   $E199
000D-   00          BRK
000E-   00          BRK
000F-   6B          ???
0010-   00          BRK
0011-   00          BRK
0012-   00          BRK
0013-   04          ???
0014-   00          BRK
0015-   FF          ???
0016-   00          BRK
0017-   01 FF       ORA   ($FF,X)
0019-   FF          ???
001A-   FF          ???
*


]
THIS DISK WILL BE COPIED



]*************************APPLE ][ FILE DEVELOPER
ID VERSION M            ** **  COPYRIGHT 1979  **
************************
CHOOSE ONE OF THE FOLLOWING OPTIONS


        <1>   COPY FILES
        <2>   CATALOG
        <3>   SPACE ON DISK
        <4>   UNLOCK FILES
        <5>   LOCK FILES
        <6>   DELETE FILES
        <7>   RESET SLOT & DRIVE
        <8>   VERIFY FILES
        <9>   QUIT
```

Fig. 12-4. Results of the execution of the XC TEXT program.

```
WHICH WOULD YOU LIKE?
                COPY FILES
SOURCE SLOT?
      DRIVE?

DESTINATION SLOT?
           DRIVE?

FILENAME?
DO YOU WANT PROMPTING?
INSERT DISKS.  PRESS <ESC> TO RETURN TO  MAIN MENU
OR ANY OTHER KEY TO BEGIN

ILENAME: =
FILE HELLO

FILE HELLO
ALREADY EXISTS.
TYPE IN A NEW FILE NAME FOR THE COPY OR
<RETURN> TO REPLACE EXISTING FILE OR
<CTRL-C><RETURN> TO CANCEL COPY
:
DONE

FILE TEXT.CHAPTER XIDFP1
DONE

FILE TEXT.CHAPTER XIDFP
DONE

FILE TEXT.CHAPTER XIDFP2
DONE
PRESS ANY KEY TO CONTINUE
*************************APPLE ][ FILE DEVELOPER
D VERSION M            **  **  COPYRIGHT 1979  **
************************
CHOOSE ONE OF THE FOLLOWING OPTIONS

        <1>   COPY FILES
        <2>   CATALOG
        <3>   SPACE ON DISK
        <4>   UNLOCK FILES
        <5>   LOCK FILES
        <6>   DELETE FILES
        <7>   RESET SLOT & DRIVE
        <8>   VERIFY FILES
        <9>   QUIT

WHICH WOULD YOU LIKE?
                QUIT
```

Fig. 12-4—*cont.* Results of the execution of the XC TEXT program.

chapter 13

Printers

Printers are used to make a permanent record of information that is stored on disk, or has been generated by an analysis of data by a program. Changing a program to print information on a printer instead of the CRT can be time consuming. Functions, such as TAB and SPC, which work perfectly well on the CRT, can cause difficulties when adapting a 40 column CRT display to a longer printline.

DEFICIENCIES OF TAB AND SPC STATEMENTS

The standard Apple II computer is designed to output 40 columns and 24 rows to a CRT (tv or monitor). There are two functions, TAB and SPC (space), to format output in the horizontal mode. The TAB function is limited to 40 spaces (0–39), while the SPC function has no limit.

Most printers have at least an 80 column capability, and many have a 163 column capability at 12 characters per inch (12 pitch).

One deficiency of the Apple II computer is the problem of writing programs in Applesoft with the TAB and SPC limitations, yet outputting to an 80 column printer.

The TAB function can certainly be used to output data to the printer but it limits the output to 40 columns. The SPC outputs to the printer, but programming every possible case is time consuming.

SOLUTION TO TAB AND SPC DEFICIENCIES

The program in Fig. 13-1 is one solution to the problem of programming in Applesoft and outputting a line length of 75 characters to the printer.

VARIABLE DICTIONARY

The variable dictionary is written directly in the program from lines 10 through 90.

Numeric input (NI) is the variable into which all numeric values are stored for output.

The field size (FS) is determined by how many columns the numeric input (NI) is going to use.

The column position (CP) is the position in which the field for outputting the value is to start. If the print string (P$) has been built to a column position of 40, and a column of 30 is specified, it will not cause the printer to back up. The program was written to add spaces to extend the print line further. The print line should be built one field at a time from left to right.

The decimal rounding (DR) variable is used to round a real number to a specific number of decimal places. When the decimal rounding variable is equal to zero (DR = 0), it means the numeric value is to be rounded to the one's place. There is no maximum or minimum limit placed on DR. If the number in NI is to the power of E + 10 or E − 10 (scientific notation), the rounding function will be bypassed.

FP$ is the string variable into which literals are placed for outputting. This is for headings or lines of explanation.

P$ is the string variable that outputs to the printer.

LP is the variable used to determine the length of FP$.

LL is the line length. In line 1000, LL is set to 75. The line length can be the same value as the line length of the printer.

SP$ is a string variable that holds 256 spaces. This is used for packing spaces into the printline and justifying the individual fields.

FJ is field justification. FJ = 0 will left justify the field. FJ = 1 will right justify the field. FJ = 2 will center justify the field.

Field type (FT) is the variable which determines if the field to be output is string or numeric type. FT = 0 indicates numeric, while FT = 1 indicates a string.

The variables J and L are used as loop variables, counters and position holders.

```
1  REM *FORMATTER SUBROUTINE
10 NI = 0: REM *NUMERIC INPUT
20 FS = 0: REM *FIELD SIZE
30 CP = 0: REM *COLUMN POSITION
40 DR = 0: REM *DECIMAL PLACES TO ROUND TO
50 FP$ = "": REM *STRING FIELD VARIABLE
60 P$ = "": REM *PRINT OUT STRING
70 LP = 0: REM *LENGTH OF FP$
80 LL = 0: REM *LINE LENGTH
90 SP$ = "          ":SP$ = SP$ + SP$:SP$ = SP$ + SP$:SP
     $ = SP$ + SP$:SP$ = SP$ + SP$:SP$ = SP$ +  LEFT$
     (SP$,127)
110 FJ = 0: REM *FIELD JUSTIFICATION(0=LEFT;1=RIGHT;2=
     CENTER)
120 FT = 0: REM *FIELD TYPE(0=NUMERIC;1=STRING)
150  GOTO 1000
200  IF FS > 0 THEN 230
210  PRINT P$:P$ = "": IF FS = 0 THEN  RETURN
220  FOR J = 1 TO  ABS (FS): PRINT : NEXT J: RETURN
230  IF  LEN (P$) < CP THEN P$ = P$ +  LEFT$ (SP$,CP -
      LEN (P$))
240  IF FT THEN 350
250  IF  ABS (NI) >  = 1E10 OR  ABS (NI) <  = 1E - 10 THEN
     FP$ =  STR$ (NI): GOTO 350
260 NI =  INT (NI * 10 ^ DR + .5) / 10 ^ DR
270 FP$ =  STR$ (NI)
280  FOR J = 1 TO  LEN (FP$)
290  IF  MID$ (FP$,J,1) = "." THEN 320
300  NEXT J:FP$ = FP$ + "."
310  IF DR < 1 THEN 350
320  IF  LEN (FP$) = J + DR THEN 350
330 J = DR + J -  LEN (FP$)
340  FOR L = 1 TO J:FP$ = FP$ + "0": NEXT L
350 LP =  LEN (FP$)
360  IF LP >  = FS THEN 420
370  ON FJ GOTO 390,400
380 FP$ = FP$ +  LEFT$ (SP$,FS - LP): GOTO 420
390 FP$ =  LEFT$ (SP$,FS - LP) + FP$: GOTO 420
395  REM
400 L =  INT ((FS - LP) / 2):J = FS - LP - L: IF L > 0
     THEN FP$ =  LEFT$ (SP$,L) + FP$
410  IF J > 0 THEN FP$ = FP$ +  LEFT$ (SP$,J)
420 P$ = P$ + FP$: IF  LEN (P$) > LL THEN  PRINT  LEFT$
     (P$,LL):P$ =  RIGHT$ (P$, LEN (P$) - LL)
430  RETURN
435  REM

1000 LL = 75
1010 D$ =  CHR$ (4): PRINT D$;"PR#1": PRINT  CHR$ (9);
     LL;"N"
1020  FOR J = 1 TO LL - 1: PRINT "*";: NEXT J: PRINT
1030 CP = 0:FT = 1:FP$ = "TEST HEADER"
1040 FS = 75:FJ = 2
1050  GOSUB 200
1060 FS =  - 2: GOSUB 200
1070 FT = 1:FS = 15:FJ = 1
```

Fig. 13-1. Printer formatter program.

```
1080  FOR M = 1 TO 4: READ FP$:CP = 15 * (M - 1)
1090  GOSUB 200: NEXT M:FS =  - 1: GOSUB 200
1100 FT = 0:FJ = 1:DR = 3
1110 AVG = 0:CNT = 0
1120  FOR M = 1 TO 2
1130 FS = 15
1140  FOR N = 1 TO 4
1150  READ D
1160 AVG = AVG + D:CNT = CNT + 1
1170 NI = D:CP = 15 * (N - 1)
1180  GOSUB 200
1190  NEXT N
1200 FS = 0: GOSUB 200
1210  NEXT M
1220  GOSUB 200
1230 FT = 1:FS = 15:FP$ = "AVERAGE=":FJ = 1:CP = 35: GOSUB
     200
1240 FT = 0:FJ = 0:DR = 0:CP = 50:NI = AVG / CNT: GOSUB
     200
1250 FS = 0: GOSUB 200
1260  PRINT D$;"PR#0"
1270  END
2500  DATA  1ST POINT,2ND POINT,3RD POINT,4TH POINT
3500  DATA  150.3427,860.2176,1021.3347,15.32
3510  DATA  106.72,75.712,89.098,569.0119
```

Fig. 13-1—*cont.* Printer formatter program.

```
RUN

**************************************************
                TEST HEADER


1ST POINT      2ND POINT      3RD POINT      4TH POINT

 150.343        860.218       1021.335         15.320
 106.720         75.712         89.098        569.012

                                        AVERAGE=361.
```

Fig. 13-2. RUN of the printer formatter program.

```
**************************************************
                TEST HEADER


 1ST POINT      2ND POINT      3RD POINT      4TH POINT

    150.34         860.22        1021.33          15.32
    106.72          75.71          89.10         569.01

                                        AVERAGE=361.0
```

Fig. 13-3. RUN of the printer formatter program with first program modification.

```
**************************************************
                TEST HEADER

1ST POINT      2ND POINT      3RD POINT      4TH POINT

150.           860.           1020.          20.
110.           80.            90.            570.

                                          AVERAGE=361.0
```

Fig. 13-4. RUN of the printer formatter program with second program modification.

PROGRAM TO CONTROL OUTPUT TO THE PRINTER

When the program, Fig. 13-1, is RUN it defaults through lines 1 to 120 to initialize the variables. Line 150 causes the program to GOTO 1000.

```
]LIST

10 D$ =  CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
     HAT IS FORTY LONG"
30  PRINT D$;"PR#1"
50  PRINT TXT$
80  PRINT D$;"PR#0"
100  END
```

```
]PR#0A LINE OF CHARACTERS THAT IS FORTY LONG


A LINE OF CHARACTERS THAT IS FORTY LONG
```

Fig. 13-5. Program and RUN on INTEGRAL DATA SYSTEMS-440 printer.

```
]LIST

10 D$ =  CHR$ (4)
30  PRINT D$;"PR#1"
70  FOR J = 1 TO 81
80  PRINT  STR$ ( INT (J / 10));
85  NEXT J
86  PRINT
90  PRINT D$;"PR#0"
100  END

]PR#0000000000111111111122222222223333333333444444444455555555556666666666777777777
77788

0000000001111111111222222222233333333334444444445555555555666666666677777777788
```

Fig. 13-6. Program and RUN on INTEGRAL DATA SYSTEMS-440 printer.

```
]LIST

 10 D$ =  CHR$ (4)
 30  PRINT D$;"PR#1"
 40  PRINT  CHR$ (9);"80N"
 70  FOR J = 1 TO 81
 80  PRINT  STR$ ( INT (J / 10));
 85  NEXT J
 86  PRINT
 90  PRINT D$;"PR#0"
 100  END
```

```
]PR#0
0000000001111111111222222222233333333334444444445555555555666666666677777777788
```

```
0000000001111111111222222222233333333334444444445555555555666666666677777777788
```

Fig. 13-7. Program and RUN on INTEGRAL DATA SYSTEMS-440 printer.

```
]LIST

10 D$ =  CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
     HAT IS FORTY LONG"
30  PRINT D$;"PR#1"
40  PRINT  CHR$ (9);"30N"
50  PRINT TXT$
80  PRINT D$;"PR#0"
100  END



]PR#0
A LINE OF CHARACTERS TH          AT IS FORTY LONG



A LINE OF CHARACTERS TH          AT IS FORTY LONG
```

Fig. 13-8. Program and RUN on INTEGRAL DATA SYSTEMS-440 printer.

MAIN PROGRAM LINES 1000–1050

At Line 1000, LL = 75, the length of the line is initialized to 75 columns. The length of the line on the printer will be 75 columns long.

```
1010 D$ = CHR$ (4): PRINT D$;"PR#1": PRINT CHR$ (9) ;LL;"N"
```

Line 1010 initializes D$, the key to DOS, turns the printer on, and initializes the printer to cause a carriage return after 75 columns have been reached. Control I, followed by 75 (CHR$(9);"75N"), is sent to the printer, and this statement turns off the CRT display. A line of 74 asterisks, Fig. 13-2, is printed out to demonstrate the length of the line.

```
1030 CP = 0:FT = 1:FP$ = "TEST HEADER"
```

Line 1030 sets the column position to zero, sets the field type to one which is a string value, and stores the literal, "TEST HEADER", in FP$. The field is to be center justified (FJ = 2).

Line 1040 initializes the field size to 75, and center justifies the field. The program jumps to the subroutine that begins at line 200. The column position should run from zero to one less than the line length (74, in this program), and the column position, plus field size, should be less than or equal to the line length (CP + FS <= LL).

FUNCTION OF FORMATTING SUBROUTINE AT 200

The subroutine at line 200 examines parameters that are set in the main program and either prints out the print string (P$) or builds values in a field onto the tail end of the print string. These parameters include what type of field justification, field size, field type, and decimal rounding if the field is numeric.

```
200 IF FS > 0 THEN 230
```

If the field size is greater than zero, it means something is to be added to P$, either a numeric value, or a string value.

If the field size is less than or equal to zero, P$ is printed, and P$ is then initialized to a null value. IF FS = 0 the program returns to line 1060.

If FS is less than zero, the third statement in line 210 (IF FS = 0 THEN RETURN) is FALSE, and the program defaults to line 220.

```
220 FOR J = 1 TO ABS (FS) : PRINT: NEXT J: RETURN
```

Line 220 causes a number of blank lines to be printed. The number of blank lines is determined by the absolute value of the field size. For example, if FS = −3, P$ is printed and three (blank) lines are skipped. If FS = 0, PRINT P$, and don't leave any blank lines. If FS is greater than zero, add a numeric or literal value to P$, and don't print anything.

```
230 IF LEN (P$) < CP THEN P$ = P$ + LEFT$ (SP$,CP-LEN(P$))
```

To get to line 230, FS >0, which means a numeric value, or literal, is to be added to P$. If the length of P$ is greater than the column position on the printer, then add as many spaces as necessary to get to the correct column position. The column position (CP) changes with every field. If additional spaces do not need to be inserted before a field is printed, that is, the boundaries of adjacent fields touch, then column position can remain unchanged.

```
240 IF FT THEN 350
```

```
]LIST
10 D$ =  CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
     HAT IS FORTY LONG"
30  PRINT D$;"PR#1"
50  PRINT TXT$
80  PRINT D$;"PR#0"
100  END
]
```

```
 RUN
A LINE OF CHARACTERS THAT IS FORTY LONG
```

Fig. 13-9. Program and RUN on CENTRONICS 737-1 printer.

```
]LIST
10 D$ =  CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
     HAT IS FORTY LONG"
30  PRINT D$;"PR#1"
40  PRINT  CHR$ (15)
50  PRINT TXT$
60  PRINT  CHR$ (14)
80  PRINT D$;"PR#0"
100  END
]
```

```
 RUN

A LINE OF CHARACTERS THAT IS FORTY LONG
```

Fig. 13-10. Program and RUN on CENTRONICS 737-1 printer.

```
]LIST
10 D$ =  CHR$ (4)
30  PRINT D$;"PR#1"
70  FOR J = 1 TO 81
80  PRINT  STR$ ( INT (J / 10));
85  NEXT J
86  PRINT
90  PRINT D$;"PR#0"
100  END
]
```

```
 RUN
000000000111111111122222222233333333344444444455555555556666666667777777778
8
```

Fig. 13-11. Program and RUN on CENTRONICS 737-1 printer.

Line 240 separates the field type into numeric and string. FT = 0 is a numeric input, and FT = 1 is a string value.

If FT = 0, a numeric value is to be added to P$, and the program defaults to line 250.

```
250 IF ABS (NI) > = 1E10 OR ABS (NI) <= 1E - 10 THEN FP$ =
    STR$ (NI) : GOTO 350
```

If the absolute value of the numeric input value is greater than or equal to 1E+10 or less than or equal to 1E–10, then the value is output in scientific notation. If the value is to be output in scientific notation the program converts NI to a string representation and jumps to line 350.

```
260 NI = INT (NI * 10 ^ DR + .5) / 10 ^ DR
```

Line 260 causes the numeric input value to be rounded.

```
270 FP$ = STR$ (NI)
```

The numeric input is converted to a string value to be able to examine it for the decimal point and the correct number of decimal places.

Lines 280 through 300 are a loop used to search the string value to

```
]LIST
10 D$ =  CHR$ (4)
30  PRINT D$;"PR#1"
40  PRINT  CHR$ (9);"80N"
70  FOR J = 1 TO 81
80  PRINT  STR$ ( INT (J / 10));
85  NEXT J
86  PRINT
90  PRINT D$;"PR#0"
100  END
]

 RUN
 000000000111111111122222222233333333344444444455555555566666666677777777778
 8
```

Fig. 13-12. Program and RUN on CENTRONICS 737-1 printer.

discover if a decimal point needs to be added. If there is no decimal point, the second statement in line 300 (FP$ = FP$ + ".") adds a decimal point.

```
310 IF DR <1 THEN 350
```

If the decimal rounding is less than one, then the value is rounded to the one's place or above (10's, 100's etc.) and no trailing zeros need to be added.

```
320 IF LEN (FP$) = J + DR THEN 350
```

If line 320 is TRUE, the numeric input has the correct number of decimal places, and no trailing zeros are added. If line 320 is false, then trailing zeros must be added.

```
330 J = DR + J - LEN (FP$)
```

Line 330 calculates the number of trailing zeros to be added. For example, FP$ = 173.4, and decimal rounding is three (DR = 3), then two trailing zeros must be added. J, in this case, is 4 (decimal point is the 4th character in FP$), and LEN (FP$) is five. (The value of J was determined in lines 280–300.)

$$J = 3(DR) + 4(J) - 5(LEN(FP\$)) = 2$$

```
]LIST
10 D$ =  CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
     HAT IS FORTY LONG"
30  PRINT D$;"PR#1"
40  PRINT  CHR$ (9);"30N"
50  PRINT TXT$
80  PRINT D$;"PR#0"
100  END
]
```

```
 RUN
A LINE OF CHARACTERS TH
AT IS FORTY LONG
```

Fig. 13-13. Program and RUN on CENTRONICS 737-1 printer.

The loop at line 340 adds the trailing zeros to FP$.

```
350 LP = LEN (FP$)
```

Line 350 is the point where the numeric value and the string value join to follow one logical path.

```
360 IF LP > = FS THEN 420
```

If the length of the literal (FP$) is greater than or equal to the field size, no field justification is necessary, and the field justification routine is bypassed. Line 420 concatenates the output value onto P$ and returns.

```
 PR#1
]PRINT CHR$(15);CHR$(27);CHR$(17)

]
```

```
 LIST
10 D$ = CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
   HAT IS FORTY LONG"
30  PRINT D$;"PR#1"
40  PRINT  CHR$ (27); CHR$ (17)
50  PRINT TXT$
80  PRINT D$;"PR#0"
100  END
]
```

```
 RUN

A LINE OF CHARACTERS THAT IS FORTY LONG
```

Fig. 13-14. Program and RUN on CENTRONICS 737-1 printer—underline turned on.

```
]LIST
10 D$ = CHR$ (4)
30 PRINT D$;"PR#1"
70 FOR J = 1 TO 81
80 PRINT STR$ ( INT (J / 10));
85 NEXT J
86 PRINT
90 PRINT D$;"PR#0"
100 END
]

 RUN
000000000111111111122222222233333333344
444444445555555555666666666677777777788
8
```

Fig. 13-15. Program and RUN on CENTRONICS 737-1 printer—underline turned on.

If line 360 is FALSE, the program defaults to the field justification routine at line 370. If the field is to be left justified (FJ = 0) the program defaults to line 380. If the field is to be right justified (FJ = 1), the program branches to line 390. If the field is to be center justified (FJ = 2), the program branches to line 400. Lines 380, 390, and 400 add blanks to the field, for the correct justification code, and concatenate that value onto P$.

```
380 FP$ = FP$ + LEFT$ (SP$,FS-LP):GOTO 420
```

If the field is to be left justified, spaces are added to the end of FP$ to fill the field size. The second statement in line 380 branches to 420 to concatenate FP$ onto the end of P$.

```
390 FP$ = LEFT$ (SP$,FS-LP) + FP$: GOTO 420
```

Line 390 right justifies the output value by adding spaces to the beginning of FP$, then branches to 420.

```
400 L = INT ((FS-LP)/2):J = FS - LP - L: IF L > 0 THEN FP$ = LEFT$ (SP$,L)
 + FP$
```

```
]LIST
10 D$ = CHR$ (4)
30 PRINT D$;"PR#1"
40 PRINT CHR$ (9)"80N"
70 FOR J = 1 TO 81
80 PRINT STR$ ( INT (J / 10));
85 NEXT J
86 PRINT
90 PRINT D$;"PR#0"
100 END
]
```

```
]RUN
0000000001111111111222222222233333333344444444445555555556666666667777777778
8
]
```

Fig. 13-16. Program and RUN on CENTRONICS 737-1 printer—underline turned on.

If the field is to be center justified, the unused portion of the field size (FS-LP) is divided in half. The correct number of spaces to be added to the end of FP$ is then calculated. For example, if FS is 15 and LP is 8, then L = INT (15 − 8)/2 = INT (7/2) = 3. Therefore, three spaces will be added to the beginning of FP$ (L = 3). J will be the number of spaces left after three spaces are added to the eight characters in FP$. J = 15 − 8 − 3 = 4. "L" on the left side = 3, and "J" on the right side = 4. If "L" is greater than zero, then add that many spaces to the beginning of FP$. If J is greater than zero (line 410), then add the spaces to the end of FP$. The length of FP$ is equal to the field size (FS), for all three justification cases.

```
420 P$ = P$ + FP$: IF LEN (P$) > LL THEN PRINT LEFT$
    (P$,LL):P$ = RIGHT$ (P$,LEN (P$)-LL)
```

Line 420 concatenates the literal (FP$) onto the print line stored in P$. If the length of P$ goes over the number of characters in the line length, the correct number of characters will be printed, and P$ will be replaced with unprinted characters, and the subroutine returns. The first time the subroutine at line 200 is called, it returns to line 1060 in the main program.

MAIN PROGRAM LINES 1060–3510

```
1060 FS = −2: GOSUB 200
```

When the field size is set to a negative value it indicates lines are to be skipped (line 220). The number of lines to be skipped causes the subroutine at 200 to output into P$ and send this line to the printer. In this case, two blank lines are skipped.

```
1070 FT = 1:FS = 15:FJ = 1
1100 FT = 0:FJ = 1: DR = 3
1230 FT = 1:FS = 15:FP$ = "AVERAGE = ":FJ = 1:CP = 35: GOSUB 200
1240 FT = 0:FJ = 0: DR = 0:CP = 50:NI = AVG / CNT: GOSUB 200
```

Lines 1070,1100,1230, and 1240 are the lines where the information is input to determine how the output is formatted. Field type, field size, field justification, decimal rounding, and column position are set in the lines.

```
1080 FOR M = 1 TO 4: READ FP$:CP = 15 * (M − 1 )
```

Four heading literals are read from data statements, and the third statement in line 1080 calculates the heading position at 0, 15, 30, and 45 to place the literals.

```
1090 GOSUB 200: NEXT M:FS = − 1: GOSUB 200
```

```
TLIT
10 D$ = CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
     HAT IS FORTY LONG"
30 PRINT D$;"PR#1"
40 PRINT CHR$ (27); CHR$ (14);
50 PRINT TXT$
80 PRINT D$;"PR#0"
100 END
]
]
RUN
A LINE OF CHARACTERS THAT IS FORTY LONG
```

Fig. 13-17. Program and RUN on CENTRONICS 737-1 printer—underline turned on and elongated print.

```
]LIST
10 D$ = CHR$ (4)
30 PRINT D$;"PR#1"
40 PRINT CHR$ (27); CHR$ (14);
70 FOR J = 1 TO 81
80 PRINT STR$ ( INT (J / 10));
85 NEXT J
86 PRINT
90 PRINT D$;"PR#0"
100 END
]
```

```
RUN
 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 4
 4444444455555555566666666677777777778

]
```

Fig. 13-18. Program and RUN on CENTRONICS 737-1 printer—underline turned on and elongated print.

```
]LIST
10 D$ = CHR$ (4)
30 PRINT D$;"PR#1"
35 PRINT CHR$ (9);"80N"
40 PRINT CHR$ (27); CHR$ (14);
70 FOR J = 1 TO 81
80 PRINT STR$ ( INT (J / 10));
85 NEXT J
86 PRINT
90 PRINT D$;"PR#0"
100 END
]
```

```
RUN
 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 4
 4444444455555555566666666677777777778
]
```

Fig. 13-19. Program and RUN on CENTRONICS 737-1 printer—underline turned on and elongated print.

Line 1090 puts the literals read in from line 1080 on P$, and closes the loop. FS = − 1 causes P$ to output one blank line by calling the subroutine at line 200.

```
1100 FT = 0:FJ = 1:DR = 3
```

Line 1100 controls the formatting of the field type. FT = 0 is a numeric field, FT = 1 is a string field. The field justification is of three types, FJ = 0 is left justified, FJ = 1 is right justified, and FJ = 2 is center justified. Decimal rounding is set to three places. See discussion of subroutine at 200, lines 250 through 340.

```
1110 AVG =0:CNT =0
```

Line 1110 initializes the value of average, and the counter, to zero. The values input are to be counted, summed, and averaged.

Lines 1120 through 1210 contain doubly nested "M" and "N" loops to read the values in the data statements, sum the values, count the number of values, and assign the values in the data statements to the numeric input (NI). These values are placed into P$ and output to the printer by calling the subroutine at line 200. The functions of the lines are as follows:

```
1120 M LOOP − NUMBER OF LINES INPUT
1130 FIELD SIZE SET TO 15 CHARACTERS
1140 READ IN FOUR NUMBERS FROM DATA STATEMENTS
1150 READ STATEMENT
1160 SUMS AND COUNTS THE NUMBERS READ IN
1170 ASSIGNS THE NUMBER READ IN TO THE NUMERIC INPUT
     (NI), AND COMPUTES COLUMN POSITION
1180 ADDS NUMERIC INPUT TO P$
1190 NEXT N
1200 SETS FIELD SIZE TO ZERO, AND PRINTS A LINE OF FOUR
     NUMBERS JUST CONSTRUCTED
1210 NEXT M

1220 GOSUB 200
```

Line 1220 prints out a blank line since FS = 0, consequently P$ is a null value (line 210).

```
1230 FT = 1:FS = 15:FP$ = "AVERAGE=":FJ = 1:CP =35: GOSUB 200
```

Line 1230 sets the field type to a string value (FT = 1), sets the field size to 15 (FS = 15). It assigns the literal to FP$ (FP$ = "average = "), sets the

field to be right justified (FJ = 1), sets the column position to 35, and calls the subroutine at line 200 to put FP$ onto P$.

```
1240 FT = 0:FJ = 0:DR = 0:CP = 50:NI = AVG / CNT: GOSUB 200
```

The field type is set to numeric (FT = 0), the field is left justified (FJ = 0), the decimal round is set to the one's place (DR = 0), the column position is set to 50. Line 1240 also assigns the average to the numeric input, and calls the subroutine at 200 to put the numeric input value onto P$.

Lines 1230 and 1240 combine to print "AVERAGE = 361.", by right justifying the literal whose field ends in position 49 and left justifying the numeric value whose field starts in column 50. By setting the borders of these two fields adjacent to one another and using the proper justification the two fields are made to appear as one.

```
1250 FS = 0: GOSUB 200
```

Line 1250 sets the field size to zero, and calls the subroutine at line 200 to print out P$ (line 420).

```
1260 PRINT D$;"PR#0"
```

```
 LIST

10 D$ =  CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
     HAT IS FORTY LONG"
30  PRINT D$;"PR#1"
50  PRINT TXT$
80  PRINT D$;"PR#0"
100  END

]
```

```
 RUN
A LINE OF CHARACTERS THAT IS FORTY LONG
```

Fig. 13-20. Program and RUN on RADIO SHACK TRS-80 DWP-410 impact printer.

Line 1260 turns the printer off and the program ends.

Fig. 13-2 was created by the RUN of the program in Fig. 13-1.

Fig. 13-3 was created by a modification of lines 1070, 1100, and 1240 in the program, Fig. 13-1.

```
1070 FT =1:FS =15: FJ =2
1100 FT =0:FJ =1: DR =2
1240 FT =0:FJ =0: DR =1:CP =50:NI =AVG / CNT: GOSUB 200
```

Fig. 13-4 was created by a modification of line 1070 in the program, Fig. 13-1.

```
1070 FT =1:FS =12:FJ = 0
```

PRINTER CONTROL

The output of three medium priced printers is presented in Figs. 13-5 through 13-23. This is not a controlled output, and no conclusions are drawn about the output or printer characteristics. The figures are presented for your evaluation.

As nearly as possible, the output was produced with the conditions constant on each printer.

The INTEGRAL DATA SYSTEMS-440 (IDS-440) is a dot matrix printer that uses a 7 by 7 dot matrix in the normal mode, and an 8 by 7 dot matrix in the enhanced mode. The CENTRONICS 737-1, standard and condensed

Chart 13-1. Comparison of Selected Printer Codes

Command	IDS-440	CENTRONICS 737	TRS-80 DWP-410
Carriage return	CHR$(13)	CHR$(13)	CHR$(13)
Line feed	CHR$(10)	CHR$(10)	---(1)
Form feed	CHR$(12)	none	none
Enhanced mode	CHR$(1)	none	none
Normal mode			
8.3 CHAR. IN.	CHR$(28)	none	none
10.0 CHAR. IN.	CHR$(29)	CHR$(27);CHR$(19)	CHR$(27);CHR$(15)
12.0 CHAR. IN.	CHR$(30)	CHR$(27);CHR$(20)	CHR$(27);CHR$(14)
16.5 CHAR. IN.	CHR$(30)	none	none
Proportional	none	CHR$(27);CHR$(17)	CHR$(27);CHR$(17)
Elongated print(start)	none	CHR$(27);CHR$(14)	none
Elongated print(stop)	none	CHR$(27);CHR$(15)	none
Start underline	none	CHR$(15)	CHR$(15)
Stop underline	none	CHR$(14)	CHR$(14)

(1) RADIO SHACK TRS-80 DWP-410 comes up with double spaced lines. CHR$(17);CHR$(21) causes the printer to single space.

mode, uses a 7 by 8 dot matrix and the elongated mode uses an "N" by 8 dot matrix. The RADIO SHACK TRS-80 DWP-410 is an impact printer.

In the normal mode (IDS-440), the standard mode (CENTRONICS), and the TRS-80, 10 characters to the inch were printed.

New ribbons were used on each printer. The quality of print can be influenced by the ink, or carbon, content of the ribbon, and the age of the ribbon.

Where possible, the same program was run on each printer. The programs run on an Apple II computer with 48K using a parallel printer interface card in slot #1.

Chart 13-1 is a list of selected control codes, used by the IDS-440, CENTRONICS 737, and the TRS-80 DWP-410. A list of control characters can be found in Fig. 1-6.

Figs. 13-5 (IDS-440), 13-9 (CENTRONICS), and 13-20 (TRS-80) are a program and run to output a line of 40 characters. The printer was turned on ("PR#1"), to list the program. After the program was listed the printer

```
]LIST

10 D$ =  CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
     HAT IS FORTY LONG"
30  PRINT D$;"PR#1"
40  PRINT  CHR$ (15)
50  PRINT TXT$
60  PRINT  CHR$ (14)
80  PRINT D$;"PR#0"
100  END

]
```

```
 RUN

A LINE OF CHARACTERS THAT IS FORTY LONG
```

Fig. 13-21. Program and RUN on RADIO SHACK TRS-80 DWP-410 impact printer.

was put offline, and the paper was scrolled up to the middle of the page. When the scrolling was complete, the printer put online, and then turned off ("PR#0"). The "PR#0" was stored in the buffer and printed out when the program was run. (The CENTRONICS and TRS-80 printers do not have a buffer.) A second run of the program produced the second line of output.

Fig. 13-9 is a listing of the same program and RUN used on the CENTRONICS printer. The CENTRONICS has no buffer to hold the output from computer memory, so the line is printed directly from computer memory. When the CENTRONICS is turned on (PR#1), and off (PR#0), the print (PR#0) is not placed on the printer paper, however the verb RUN is placed on the printer paper. The program when RUN turns the printer on and off using the deferred mode.

The CENTRONICS 737-1 and the TRS-80 DWP-410 have similar control codes. Both printers are based on the CENTRONICS design. However, the CENTRONICS 737 has a 40 pin female connector plug, while the TRS-80 DWP-410 has a 36 pin female connector plug.

```
]LIST

10 D$ =  CHR$ (4)
20 TXT$ = "A LINE OF CHARACTERS T
     HAT IS FORTY LONG"
30  PRINT D$;"PR#1"
40  PRINT  CHR$ (9);"30N"
50  PRINT TXT$
80  PRINT D$;"PR#0"
100  END

]


 RUN

A LINE OF CHARACTERS THAT IS FORTY LONG
```

Fig. 13-22. Program and RUN on RADIO SHACK TRS-80 DWP-410 impact printer.

```
]LIST
 10 D$ =  CHR$ (4)
 30  PRINT D$;"PR#1"
 40  PRINT  CHR$ (9);"70N"
 50  FOR J = 1 TO 81
 60  PRINT  STR$ ( INT (J / 10));
 70  NEXT J
 75  PRINT
 80  PRINT D$;"PR#0"
100  END
]
RUN
000000000111111111222222222333333334444444455555555566666666
667777777778
```

Fig. 13-23. Program and RUN on RADIO SHACK TRS-80 DWP-410 impact printer—"70N" does not affect output.

The CENTRONICS and TRS-80 have a very useful feature that the IDS-440 does not have. The CENTRONICS and TRS-80 have an underline feature, Figs. 13-10, and 13-21. PRINT CHR$(15) turns the underline on, and PRINT CHR$(14) turns the underline off.

CENTRONICS, Figs. 13-14 through 13-18, output should be self-explanatory because the control code is at the beginning of the figure.

TRS-80 DWP-410, Fig. 13-23, produces an output of 80 characters, even though CHR$(9);"70N" should limit output to 70 characters. No explanation can be given.

Printers in the medium price range ($1300–$1800 plus interface card) can vary greatly in quality and standard features. Before a printer is purchased, the user should know what specific requirements are needed for the average job. By studying the owner's manual, the specific characteristics can be discovered. By combining data about job requirements and printer characteristics, the best printer value can be purchased.

appendix

Converting Decimal to Hexadecimal and Converting Hexadecimal to Decimal

Hexadecimal is a number system whose base, or radix, is 16, Table A-1. Decimal is a number system whose base is 10, octal is a number system whose base is 8, and binary is a number system whose base is 2. The two numeric characters in the binary system are one (1) and zero (0). At an early age most Americans learn the decimal system. All computers, including the Apple, use the binary system for all mathematical and logical functions. Binary is too inefficient for most humans to deal with, therefore, computers display the hexadecimal or octal number system. The Apple computer displays hexadecimal for addressing, mathematical, and logical functions. The Apple computer converts binary to hexadecimal for display purposes only.

The computer user should learn to think in hexadecimal to understand the computer, so decimal must be converted to hexadecimal and hexadecimal must be converted to decimal.

Table A-1. Number Systems

16^4	16^3	16^2	16^1	16^0
65536	4096	256	16	1

10^6	10^5	10^4	10^3	10^2	10^1	10^0
1000000	100000	10000	1000	100	10	1

8^7	8^6	8^5	8^4	8^3	8^2	8^1	8^0
2097152	262144	32767	4096	512	64	8	1

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
2048	1024	512	256	128	64	32	16	8	4	2	1

The hexadecimal base contains 16 numbers. The first 10 numbers are zero (0) to nine (9), and the remaining six numbers are represented by the alpha characters "A" through "F." A = 10, B = 11, C = 12, D = 13, E = 14, and F = 15, Table A-2.

The maximum decimal value that is an acceptable address on the Apple computer with 64K is 65535. The decimal value 65535 is represented in hexadecimal as FFFF. Hexadecimal value in relation to 64K RAM is repre-

Table A-2. First 16 Hexadecimal Digits

Binary	Hexadecimal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

sented by a maximum of four characters. The four characters range from "0000" to "FFFF."

The easiest way to convert hexadecimal to decimal is to use a conversion table, Table A-3, or a calculator specifically designed for conversion from one system to another.

Table A-3 contains five columns. The first column contains the hexadecimal numeric and alpha characters from one to nine and "A" (10) through "F" (15). The second column represents the hexadecimal values of 16 cubed (16^3), multiplied by the hexadecimal value in column one. The third column represents the hexadecimal value of 16 squared (16^2), times the hexadecimal value in column one. The fourth column is the hexadecimal value of 16 (16^1), multiplied by the hexadecimal value in column one. The fifth column is the hexadecimal value of one (16^0), multiplied by the hexadecimal value in column one. To convert "FFFF" to decimal, the value in column five (15) is added to the value in column four (240), these values are added to the value in column three (3840), and this is added to the value in column two (61440). The total value of "FFFF" is 65535.

Conversion from decimal to hexadecimal is a straightforward process. The decimal number is divided by 16, and the remainder is the hexadecimal number for that position. The remainder from the first division is the hexadecimal number in the most right hand position. The decimal number to be converted to hexadecimal is "876."

Table A-3. Conversion Table

Hex X	A $X*16^3$	B $X*16^2$	C $X*16^1$	D $X*16^0$
1	4096	256	16	1
2	8192	512	32	2
3	12288	768	48	3
4	16384	1024	64	4
5	20480	1280	80	5
6	24576	1536	96	6
7	28672	1792	112	7
8	32768	2048	128	8
9	36864	2304	144	9
(10) A	40960	2560	160	10
(11) B	45056	2816	176	11
(12) C	49152	3072	192	12
(13) D	53248	3328	208	13
(14) E	57344	3584	224	14
(15) F	61440	3840	240	15

1. Divide 876/16 = 54(864) Remainder = 12
2. Divide 54/16 = 3(48) Remainder = 6
3. Divide 3/16 = 0(0) Remainder = 3

Decimal 876 is hexadecimal 36C

To convert hexadecimal to decimal multiply the hexadecimal number by the positional power, and add the decimal numbers produced by the multiplication.

	Positional Power		
	16^2	16^1	16^0
	HEX 36C		
		C = 12 * 1 =	12
		6 * 16 (16^1) =	96
		3 * 256 (16^2) =	768
Sum			876

Index

SAMS APPLE® BOOKS

Many thanks for your interest in this Sams Book about Apple II® microcomputing. Here are a few more Apple-oriented Sams products we think you'll like:

POLISHING YOUR APPLE®, Vol. 1

Clearly written, highly practical, concise assembly of all procedures needed for writing, disk-filing, and printing programs with an Apple II. Positively ends your searches through endless manuals to find the routine you need! By Herbert M. Honig. 80 pages, 5½ x 8½, comb. ISBN 0-672-22026-1. © 1982.
Ask for No. 22026 ..... **$4.95**

POLISHING YOUR APPLE®, Vol. 2

A second Apple II timesaver that guides intermediate-level programmers in setting up professional-looking menus, using effective error trapping, and making programs that run without the need for detailed explanations. Includes many sample routines. By Herbert M. Honig. 112 pages, 5½ x 8½, soft. ISBN 0-672-22160-8. © 1983.
Ask for No. 22160 ..... **$4.95**

APPLESOFT LANGUAGE (2nd Edition)

Quickly introduces you to Applesoft syntax and programming, including advanced techniques, graphics, color commands, sorts, searches, and more! New material covers disk operations, numbers, and number programming. Many usable routines and programs included. By Brian D. Blackwood and George H. Blackwood. 274 pages, 6 x 9, comb. ISBN 0-672-22073-3. © 1983.
Ask for No. 22073 ..... **$13.95**

APPLE® II APPLICATIONS

Presents a series of board-level interfacing applications you can modify if necessary to help you use an Apple II as a development system, a data-acquisition or control device, or for making measurements. Includes programs. By Marvin L. De Jong. 256 pages, 5½ x 8½, soft. ISBN 0-672-22035-0. © 1983.
Ask for No. 22035 ..... **$15.95**

DISKS, FILES, AND PRINTERS FOR THE APPLE® II

Provides you with basic-to-advanced details for using disks, files, and printers with an Apple II, including outstanding explanations for programming with sequential-access, random-access, and executive files. By Brian D. Blackwood and George H. Blackwood. 216 pages, 6 x 9, comb. ISBN 0-672-22163-2. © 1983.
Ask for No. 22163 ..... **$15.95**

THE APPLE® II CIRCUIT DESCRIPTION

Provides you with a detailed circuit description for all revisions of the Apple II and Apple II+ motherboard, including the keyboard and power supply. Highly valuable data that includes timing diagrams for major signals, and more. By Winston D. Gayler. 176 pages plus foldouts, 8½ x 11, comb. ISBN 0-672-21959-X. © 1983.
Ask for No. 21959 ..... **$22.95**

INTERMEDIATE LEVEL APPLE® II HANDBOOK

Provides you with a nicely paced transition from Integer BASIC into machine- and assembly-language programming with the Apple II. Covers text display, video POKEs, graphics, using machine language with BASIC, memory addresses, debugging, and more. By David L. Heiserman. 328 pages, 6 x 9, comb. ISBN 0-672-21889-5. © 1983.
Ask for No. 21889 ..... **$16.95**

APPLE® FORTRAN

Only fully detailed Apple FORTRAN manual on the market! Ideal for Apple programmers of all skill levels who want to try FORTRAN in a business or scientific program. Many ready-to-run programs provided. By Brian D. Blackwood and George H. Blackwood. 240 pages, 6 x 9, comb. ISBN 0-672-21911-5. © 1982.
Ask for No. 21911 ..... **$14.95**

APPLE® II ASSEMBLY LANGUAGE

Shows you how to use the 3-character, 56-word vocabulary of Apple's 6502 to create powerful, fast-acting programs! For beginners or those with little or no assembly language programming experience. By Marvin L. De Jong. 336 pages, 5½ x 8½, soft. ISBN 0-672-21894-1. © 1982.

ENHANCING YOUR APPLE® II — Vol. 1

Shows you how to mix text, LORES, and HIRES anywhere on the screen, how to open up whole new worlds of 3-D graphics and special effects with a one-wire modification, and more. Tested goodies from a trusted Sams author! By Don Lancaster. 232 pages, 8½ x 11, soft. ISBN 0-672-21846-1. © 1982.
Ask for No. 21846 ..... **$17.95**

CIRCUIT DESIGN PROGRAMS FOR THE APPLE® II

Programs quickly display "what happens if" and "what's needed when" as they apply to periodic waveform, rms and average values, design of matching pads, attenuators, and heat sinks, solution of simultaneous equations, and more. By Howard M. Berlin. 136 pages, 8½ x 11, comb. ISBN 0-672-21863-1. © 1982.
Ask for No. 21863 ..... **$15.95**

TO THE READER

Sams Computer books cover Fundamentals — Programming — Interfacing — Technology written to meet the needs of computer engineers, professionals, scientists, technicians, students, educators, business owners, personal computerists and home hobbyists.

Our Tradition is to meet your needs and in so doing we invite you to tell us what your needs and interests are by completing the following:

1. I need books on the following topics:

_____
_____
_____
_____
_____
_____

2. I have the following Sams titles:

_____
_____
_____
_____
_____
_____

3. My occupation is:

_____ Scientist, Engineer
_____ Personal computerist
_____ Technician, Serviceman
_____ Educator
_____ Student
_____ D P Professional
_____ Business owner
_____ Computer store owner
_____ Home hobbyist
Other _____
_____

Name (print) _____

Address _____

City _____ State _____ Zip _____

Mail to: **Howard W. Sams & Co., Inc.**
Marketing Dept. #CBS1/80
4300 W. 62nd St., P.O. Box 7092
Indianapolis, Indiana 46206

22163

HARDWARE CROSS REFERENCE

Company		CPU				Coprocessor				I/O				Peripherals				BUS								SYSTEM TYPE								
		6809	68000	68020	68030	68881	68451	68851	Other	SIO	PIO	NET	Other	F.D.	Hard	Tape	Other	VME	MB1	MB2	STD	G-64	PC/XT	Prop.	SBC	Other	S/W Dev.	Graph	Proc. Ctrl.	Speech	Com.	Inst.	Vision	Bus.	Other
AAA Chicago Computer Center	Page 5	x	x	x		x				x	x			x	x								x	x		x		x			x	x		x	
Artificial Intelligence S.r.l.	8		x	x	x	x	x	x		x	x	x		x	x	x		x									x	x	x						
Bernecker & Rainer GmbH	9		x			x				x	x	x	PLC	x	x	x	SCSI								x		x	x	x		x	x			
BICC Vero Microsystems	11		x	x		x	x	x		x	x	x	GPIB, SCSI	x	x	x		x									x	x	x	x	x	x	x		Industrial Automation
Cambridge Micro Computers Limited	17		x	x	x	x		x	68882	x	x	x		x	x	x		x									x	x	x		x	x		x	
Cambridge Microprocessor Systems Ltd.	19		x							x	x		Graphics	x	x			x							x			x	x		x				Music
Ciprico, Inc.	21														x	x		x	x	x															
Comendec Ltd.	22											x						x																	
Communication Machinery Corporation	23		x															x	x				x												
Compcontrol B.V.	24	x	x	x	x	x	x	x		x	x	x	Industrial	x	x	x	Optical	x							x		x		x		x	x	x		x
Cyclone Microsystems	29			x	x	x		x		x	x			x	x			x									x	x	x	x	x	x	x	x	
Data-Comp Division - CPI	30	x	x	x		x				x	x	x		x	x	x								x	x		x		x		x	x		x	
Datacube	32		x	x						x				x	x			x										x				x			Imaging
Datel, Inc.	35												Analog/TTL					x	x																
Digalog Systems, Inc.	36		x							x	x	x	EM Bus	x	x	x						x													DSP ATE
Digital Electronics Corporation	37	x								x	x			x	x			x									x								
Dipl. Phys. M. K. System Forschung	38		x	x		x				x	x	x		x	x	x		x									x	x	x		x	x			
Dorsch Microsystem GmbH	41		x	x		x		x		x	x	x	Analog Video	x	x	x		x									x	x		x	x				
Dr. Rudolf Keil GmbH	44		x									x						x							x										
EKF Elektronik GmbH	45	x	x							x	x	x	GIPB	x	x	x		x								x	x	x	x	x	x	x			
Epstein Associates	49		x	x		x	x	x		x	x		x	x	x			x							x		x		x	x	x	x			
Fairlight Instruments	50	x	x	x						x	x			x	x	x	WORM							x											Music
Fujifacom Corporation	51		x							x	x			x	x									x					x						
General Micro Systems, Inc.	52		x	x	x	x				x	x			x	x			x								x	x								
Gespac SA	55	x	x	x	x	x				x	x	x		x	x	x						x					x	x	x		x	x	x		Motion Control
Graphic Strategies	60																Graphic	x																	
GWK Technische Elektronik GmbH	61		x	x	x	x				x	x	x	Process I/O	x	x	x	Graphic	x						x			x	x	x		x	x	x	x	
H. C. Andersen Computer A/S	62	x								x	x		x	x	x	x									x		x	x	x	x	x			x	
Hazelwood Computer Systems	63		x	x		x				x	x	x		x	x	x	x								x		x	x	x		x	x		x	
Heurikon Corporation	68		x	x	x	x	x	x		x	x	x		x	x	x		x									x		x				x		
INCAA Computers BV	70		x							x				x	x	x	SCSI							x			x		x		x			x	
Innovision Corporation	71		x	x	x	x		x		x		x		x	x	x		x									x	x	x			x	x		
Introl Corporation	72														x			x																	
LP Elektronik GmbH	73		x						82768	x	x	x		x	x								x			x	x	x	x		x				
Matric Limited	76		x							x	x	x	Industrial	x	x									x					x		x	x			x

HARDWARE CROSS REFERENCE

Company		CPU				Coprocessor				I/O				Peripherals				BUS								SYSTEM TYPE								
		6809	68000	68020	68030	68881	68451	68851	Other	SIO	PIO	NET	Other	F.D.	Hard	Tape	Other	VME	MB1	MB2	STD	G-64	PC/XT	Prop.	SBC	Other	S/W Dev.	Graph	Proc. Ctrl.	Speech	Com.	Inst.	Vision	Bus.	Other
Matrix	Page 78	x	x							x	x		x	x	x			x			x						x		x		x	x			
Michels & Kleberhoff Computer GmbH	83		x	x		x				x	x	x		x	x	x									x		x	x	x	x	x	x	x	
Microboards	85		x	x	x	x			x	x	x	x		x	x	x		x																
Microproject Corporation	89		x	x		x				x	x	x		x	x	x		x									x		x			x	x		
Micro Concepts	91		x						82786	x	x			x	x										x		x	x							
Microsys Electronics GmbH	92		x	x	x	x	x	x		x	x	x		x	x	x		x						x			x	x	x		x	x			
Microtech Electronics Ltd.	97		x							x	x			x	x		Graphic								x				x			x			x
M.I.I.	98	x	x			x				x	x	x		x	x	x						x					x		x		x	x			
Mizar	101		x	x		x	x	x		x	x			x	x			x									x		x						
Pan Controls Ltd.	104												Servo									x													
PEP ModularComputers GmbH	105	x	x	x		x				x	x	x		x	x	x		x									x	x	x		x	x			
Powerframe	109		x	x		x		x		x	x	x		x	x	x		x									x	x		x	x		x	
PSI Systems Ltd.	110		x			x				x	x		Graphics	x	x			x									x	x	x						
Radstone Technology	112		x	x	x	x		x		x	x		SCSI	x	x	x		x									x		x	x	x				
Robcon OY	116	x	x	x		x		x		x								x								x									
Scorpion Technologies	117			x	x	x		x				x		x	x	x							x				x	x	x	x	x	x	x		
Seiko Instruments U.S.A.	118		x							x	x			x	x			x											x			x		Robotics	
Syntel Microsystems	119	x	x	x		x				x	x		Ethernet	x	x	x		x									x	x	x		x	x	x		
Themis/Tekelec - Airtronic	123		x	x		x				x	x	x		x	x	x		x				x					x	x						
TJP Electronics Ltd.	127		x	x		x					x	x		x	x			x				x					x	x	x					
Ultrascience	129		x	x	x	x		x		x	x	x		x	x	x	Optical	x					x				x	x	x		x			x	
VME Specialists	134		x	x		x				x	x		SCSI	x	x			x									x		x		x	x			
Windrush Micro Systems Ltd.	137	x	x	x	x	x			x	x	x	x	A/D	x	x	x	Optical					x			x	STE	x	x	x			x			x
Wordsworth Technology Ltd.	141		x	x		x				x	x	x		x	x	x		x									x	x	x		x	x			
Xycom	144		x	x		x				x	x			x	x			x									x		x		x	x	x		
XYZ Electronics, Inc.	148	x	x							x	x		Video	x	x						x						x								

DISKS, FILES, AND PRINTERS FOR THE APPLE® II

Learn how to create and use text files while running a computer program, and to make hard copies of the results on a printer. Covers Apple II, II+, IIe, and Franklin Microcomputers.

Students of any age will discover:

- Using the system includes booting DOS master, initializing disk, backup disk, accessing the drives, verifying, saving programs, and loading programs.
- DOS commands enable you to operate the disk drives and printer from the keyboard and to change languages (Integer BASIC to Applesoft or Applesoft to Integer BASIC).
- Sequential files include creating, updating, reading, extending, refining, and multisection.
- Random access files are faster and take less memory.
- Executive files can be used to control a series of steps that do not require the operator to make keyboard inputs.
- Deficiencies of TAB and SPC can be overcome in 80 characters per line printers.

HOWARD W. SAMS & CO., INC.
4300 West 62nd Street, Indianapolis, Indiana 46268 USA

$15.95/22163

ISBN: 0-672-22163-2